# L'AMSTRAD EXPLORÉ

JOHN BRAGA




L'AMSTRAD
EXPLORÉ
JOHN BRAGA




JOHN BRAGA




Paris • Berkeley • Düsseldorf

Traduction française de Henry-Luc Planchat

CP/M est une marque déposée de Digital Research Inc.
WordStar est une marque déposée de Micropro Inc.



ISBN 2-7361-0135-4
(Edition originale ISBN 0-7457-0131-0 Kuma Computers Ltd.)

# PRÉFACE

L'Amstrad CPC464 est l'un des ordinateurs familiaux les plus intéressants parmi ceux qui ont été lancés en 1984. Il a déjà été largement adopté par ceux qui recherchent une machine proposant un langage et un système d'exploitation particulièrement performants ainsi qu'un excellent rapport qualité-prix. Par nature, les micro-ordinateurs sont des appareils compliqués et les esprits curieux ne tardent pas à se demander comment fonctionnent leurs différents éléments. Cela n'est jamais très facile à comprendre et nous espérons que ce livre répondra à la plupart des questions qui se posent immédiatement, tout en montrant comment s'est déroulée notre propre "exploration".

TIM MOORE
septembre 1984,
Kuma Computers Ltd.

# S O M M A I R E

Préface ........................................ 5
Introduction ........................................ 9

## I
## SE FAMILIARISER AVEC LA MACHINE

1. Les possibilités de l'Amstrad ........................ 12
2. Une visite guidée en BASIC ........................ 18
3. Touches et caractères ........................ 32
4. La splendeur du technicolor ........................ 42
5. Texte et graphisme ........................ 49
6. Hi-fi programmable ! ........................ 62

## II
## UN SYNTHÉTISEUR MUSICAL TRÈS COMPLET

7. La notation musicale ........................ 68
8. Volume et hauteur d'un son ........................ 72
9. Harmonie à deux et trois voix ........................ 77
10. Effets spéciaux ........................ 88

## III
## LES POSSIBILITÉS GRAPHIQUES

11. Animation et illusion ........................ 92
12. Jeu n° 1 — Le Heffalump affamé ........................ 101
13. Jeu n° 2 — Les fantômes dans la bouteille ! ........................ 110

## IV
## PROGRAMMATION EN LANGAGE D'ASSEMBLAGE

14. L'environnement ZEN .............................. 120
15. Interface avec le BASIC .............................. 125
16. Les routines du système d'exploitation ............... 136

# INTRODUCTION

Bienvenue dans le monde de l'Amstrad CPC464 ! Ce système puissant et passionnant a reçu de nombreuses critiques favorables depuis sa première apparition au cours de l'année 1984. Ce n'est pas étonnant, car il offre un excellent rapport qualité-prix, et procure de nombreuses possibilités sous une présentation solide et soignée.

Ce livre s'adresse à tous les possesseurs de l'Amstrad et à ses utilisateurs potentiels, quel que soit leur niveau. Il a été conçu pour augmenter les informations fournies dans le manuel. Mais plutôt que reproduire simplement les données du manuel, ce livre s'attarde sur certains des aspects les plus exceptionnels du système, et en particulier sur deux de ses traits caractéristiques : le graphisme et le son. De nombreux exemples de programmes sont donnés pour permettre au lecteur de tester lui-même les techniques concernant ces aspects.

L'intérêt et la valeur d'un ordinateur familial résident principalement dans sa capacité d'augmenter les connaissances et le talent de l'utilisateur. Il n'est pas intéressant de copier de nombreux programmes sans comprendre quelle était l'intention de l'auteur. Par conséquent, ce livre veut expliquer *pourquoi* les choses fonctionnent, et pas simplement ce qu'il faut taper. Cela devrait permettre au lecteur de faire des ajouts aux programmes décrits dans le livre, d'écrire ses propres jeux et d'améliorer les exemples donnés. L'auteur pense que l'on ne doit pas renier les jeux, car ils peuvent se révéler particulièrement éducatifs si le lecteur se donne la peine de regarder derrière la façade.

JOHN BRAGA,
septembre 1984.

I

# SE FAMILIARISER AVEC LA MACHINE

Cette partie fait le point sur les diverses caractéristiques de l'Amstrad, et doit être considérée comme nécessaire pour bien comprendre les parties suivantes.

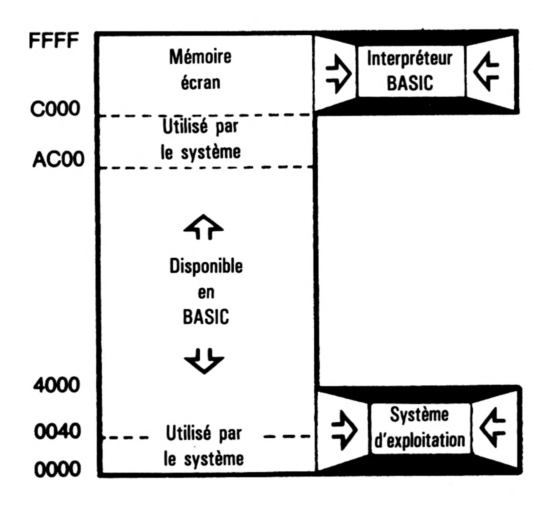
# 1
# LES POSSIBILITÉS DE L'AMSTRAD

Dans ce chapitre, nous donnerons une vue d'ensemble du système Amstrad CPC464 et nous présenterons les éléments qui seront décrits plus précisément par la suite.

## UNE MÉMOIRE D'ÉLÉPHANT !

L'Amstrad a une mémoire généreuse. Les jours sont bien lointains où les ordinateurs familiaux devaient se contenter d'1K ou de 2K de RAM. Vous pouvez maintenant profiter de 64K. Cela ne signifie peut-être pas grand-chose en soi. Il n'est pas rare qu'un système prétendant disposer de 64K n'offre en fait qu'une mémoire considérablement plus réduite à l'utilisateur lorsque l'ordinateur a rempli toute la partie de la RAM qui lui est nécessaire pour sa mise en route ! Ce n'est pas le cas de celui-ci, car l'utilisateur peut disposer de 42K en BASIC, ce qui est bien suffisant pour des programmes "à l'échelle humaine", comme nous le verrons. Certains systèmes omettent sournoisement de préciser qu'une bonne partie de la précieuse mémoire est accaparée par la gestion de l'écran lors de l'utilisation du graphisme haute résolution. Mais l'Amstrad utilise constamment 16K pour l'écran, quelle que soit la résolution graphique, et la mémoire n'est donc pas diminuée lorsqu'on choisit la haute résolution.

En fait, vous disposez de plus de 64K, car il y a en plus deux ROM de 16K (ou une ROM de 32K, pour être plus précis) qui contiennent le système d'exploitation et l'interpréteur BASIC proprement dit. Comme le Z80 ne peut adresser que 64K, une technique particulière permet aux ROM de pouvoir "apparaître" ou "disparaître" à volonté. La répartition de la mémoire est la suivante :




Normalement, cet "appel" des ROM est tout à fait invisible pour qui programme en BASIC. Quand vous désirez lire une adresse (avec PEEK), les ROM sont masquées et vous ne voyez que la RAM.

## PLUS QU'UN SIMPLE BASIC...

Le BASIC est un langage aisément compréhensible, qui possède de nombreuses caractéristiques intéressantes. En plus des possibilités auxquelles vous pouvez normalement vous attendre, le BASIC de l'Amstrad est "étendu" afin de gérer le graphisme, le son, et de permettre également une utilisation multitâche. Contrairement à d'autres systèmes, qui obligent les programmeurs à recourir à de nombreuses instructions PEEK et POKE dès qu'ils veulent réaliser quelque chose d'intéressant, le BASIC de l'Amstrad est remarquablement lisible.

Le BASIC de l'Amstrad est un BASIC complet ; il n'est donc pas nécessaire de rajouter des extensions particulières pour accéder à certains types de commandes. En conséquence, il ne vous sera que très rarement nécessaire d'utiliser l'assembleur pour bénéficier de tous les avantages du matériel — presque tous peuvent être exploités grâce au langage BASIC. Cela fait de l'Amstrad un système très facile à utiliser par quelqu'un qui n'est pas encore très familiarisé avec l'informatique.

Et il est rapide. Assez rapide pour que l'on puisse programmer directement de nombreux jeux, même ceux qui requièrent une animation graphique — mais bien entendu, un emploi judicieux de sous-programmes en assembleur peut permettre d'augmenter encore la vitesse. L'assembleur est traité dans la quatrième partie, et les jeux de la troisième partie contiennent des sous-programmes en assembleur permettant d'améliorer la gestion de l'écran.

## AVEZ-VOUS DÉJÀ PROGRAMMÉ ?

Nous avons déjà dit que l'Amstrad était un excellent ordinateur pour les nouveaux venus à l'informatique. Mais le programmeur expérimenté n'a aucune raison de se sentir lésé. L'Amstrad peut paraître inhabituel en ce sens que son manuel technique explique tous les détails matériels de la machine. Il décrit assez bien le système d'exploitation et le véritable enthousiaste se doit de le lire sérieusement. En fait, il est remarquablement facile de programmer l'Amstrad en code machine car on peut utiliser à tout moment les fonctions offertes par le système. Cela facilitera grandement la tâche des éditeurs de logiciels qui désireront éditer des programmes de traitement de texte ou des tableurs.

## UN CLAVIER TRÈS SOUPLE

Le clavier a une disposition commerciale standard, il est d'une utilisation agréable durant de longues périodes et il possède un pavé numérique complet. Si vous n'aimez pas l'agencement des touches, vous pouvez le modifier car la valeur de chaque touche peut être "reprogrammée". Vous pouvez même supprimer ou changer la vitesse de la répétition automatique. Vous n'êtes pas non plus limité par le large assortiment de caractères disponibles, puisque vous pouvez aisément créer vos propres caractères — quelques monstres pour un jeu d'arcade ou même tout un nouvel alphabet si vous le désirez. Nous en reparlerons plus tard.

Une simple touche peut également être utilisée pour afficher une chaîne de plusieurs caractères en redéfinissant une touche de fonction.

## LES CASSETTES SONT BONNES POUR LA HI-FI...

La plupart des utilisateurs de systèmes à cassettes s'efforcent de passer aux disquettes dès qu'ils en ont l'occasion. En vérité, les cassettes ne sont pas idéales pour conserver les informations, car elles sont très lentes et peu fiables en comparaison des disques souples. Mais le lecteur de cassettes de l'Amstrad est intégré à la machine et nous n'avons eu aucun problème d'erreur de lecture. Il n'y a pas à s'occuper du réglage de la tonalité, et le contrôle du volume vous permet simplement d'écouter la lecture ou l'écriture du programme si vous le désirez. L'absence de câbles, le fait qu'il n'y ait pas d'unités séparées, tout cela est un avantage certain. Pour la plupart des utilisateurs, un unique câble d'alimentation sera relié au moniteur (couleur ou monochrome) de l'ordinateur. Simplicité !

Une autre caractéristique utile de l'Amstrad : l'utilisateur peut sauvegarder ses programmes à vitesse double s'il le désire — à 2 000 bps au lieu de 1 000 bps. Il est vrai que le risque d'erreur augmente à grande vitesse, mais en employant des cassettes de bonne qualité (C12 ou C30 de préférence), il ne devrait pas y avoir de problèmes. Nous avons toujours utilisé la grande vitesse dans des conditions normales et la vitesse lente pour effectuer des copies.

## ET L'ÉCRAN ?

Les fenêtres (sur l'écran, bien sûr !) sont très à la mode actuellement en informatique. Pour le texte, l'utilisateur peut avoir jusqu'à huit fenêtres, qui peuvent se superposer de n'importe quelle manière. De plus, il dispose d'une fenêtre graphique indépendante qui peut couvrir l'ensemble de l'écran s'il le désire. Il est bon de noter qu'il n'y a pas de restrictions au mixage du texte et du graphisme dans une zone donnée. En règle générale, le graphisme est tel qu'il permet des jeux très élaborés, et des applications sérieuses peuvent être rendues beaucoup plus attrayantes si l'on emploie astucieusement les couleurs.

A propos des couleurs, l'Amstrad en offre 27, bien qu'on ne puisse pas toutes les afficher simultanément sur l'écran. Il y a une relation entre l'éventail des couleurs et le degré de résolution. Si vous choisissez d'afficher 80 colonnes sur l'écran (haute résolution), vous ne disposerez que de 2 couleurs ; par contre, si vous réduisez la résolution à 20 colonnes, vous disposerez de 16 couleurs sur l'écran.

Pour la plupart des utilisateurs, le mode intermédiaire constituera un compromis satisfaisant : 4 couleurs et 40 colonnes.

Les trois modes permettent d'afficher 25 lignes.

## UN PAYS DE COCAGNE SONORE !

L'Amstrad, sans doute conscient de la spécialité de sa compagnie, possède d'impressionnantes possibilités sonores. Il dispose de trois canaux sonores et d'un canal de bruit, qui peuvent être manipulés et synchronisés pour produire les meilleurs résultats. Une fois encore, on peut accéder à toutes ces possibilités à partir du BASIC, mais les commandes SOUND, ENVELOPPE et ENT sont très complexes et il n'est pas facile d'obtenir les effets désirés si l'on n'a pas étudié ces commandes de manière approfondie. C'est pourquoi la deuxième partie a pour objet de montrer ce qui peut être réalisé dans ce domaine.

D'autres ordinateurs familiaux offrent également d'intéressantes possibilités sonores, mais celles-ci sont limitées par l'emploi d'un minuscule haut-parleur interne. L'Amstrad dispose d'une sortie qui peut être connectée à un système hi-fi stéréo, ce qui rend les effets sonores beaucoup plus impressionnants. Mais bien entendu vous pouvez utiliser le haut-parleur interne si vous le désirez.

## ET ENSUITE ?

La plupart des utilisateurs voudront étendre leur système. Pour beaucoup, le premier choix se portera sans doute sur une imprimante, et le système peut être relié à une unité possédant une sortie industrielle standard du type "Centronics parallèle" qui permet la compatibilité avec un large éventail de machines. Pour les joueurs, les manettes de jeu seront particulièrement appréciées. Pour les utilisateurs plus sérieux, le lecteur de disquette constituera une aubaine inestimable qui ajoutera beaucoup à la vitesse et à la flexibilité du système. L'Amstrad pourra alors profiter des logiciels utilisables sous CP/M.

Une autre solution consiste à ajouter des logiciels spécialisés sous forme de ROM. L'Amstrad est conçu pour permettre l'ajout de 240 (!) ROM supplémentaires de 16K chacune, grâce à des cartes d'extension. Ces ROM d'expansion occupent toutes le même champ

d'adresse et le système n'en utilise qu'une seule à la fois, les autres étant "invisibles". Elles se sont révélées très populaires avec le système Acorn BBC et l'on peut espérer que l'Amstrad disposera très prochainement, grâce à cette méthode, de tableurs, de traitements de texte, et d'autres langages, tels que le Pascal et le Forth. Amstrad a déjà publié un guide sur la manière de créer des ROM pour le système CPC464 ; c'est un signe encourageant.

En examinant la figure décrivant l'implantation de la mémoire, présentée un peu plus tôt dans ce chapitre, vous verrez que la partie supérieure de la mémoire est occupée par une ROM de 16K contenant le BASIC. C'est cette ROM qui peut être "recouverte" par une ROM d'expansion. La ROM inférieure, contenant le système d'exploitation, ne doit jamais être "recouverte".

Vivement le Pascal !

## RÉSUMÉ

Comme vous avez pu le constater, nous avons été vivement impressionnés par le soin considérable apporté à la conception de l'Amstrad CPC464. Les possibilités offertes récompensent largement les efforts d'une étude approfondie, que nous allons entamer maintenant...

# 2

# UNE VISITE GUIDÉE EN BASIC

Le but de ce livre n'est pas de vous enseigner le BASIC. Il existe des douzaines d'ouvrages qui se proposent de le faire. Son but n'est même pas de vous apprendre les instructions du BASIC de l'Amstrad, car pour cela vous devez étudier le manuel fourni avec l'appareil. Dans ce chapitre, nous considérerons que vous comprenez au moins les rudiments du BASIC, et nous nous pencherons donc sur certaines des caractéristiques les plus inhabituelles ou les plus complexes.

Les commandes du son et du graphisme seront plutôt détaillées dans des chapitres ultérieurs.

La version du BASIC la plus commune est le MBASIC de Microsoft, que l'on trouve sur divers systèmes 8 bits ou 16 bits. Le BASIC de l'Amstrad, développé par Locomotive Software, est très complet et possède de nombreuses similitudes avec le MBASIC ; vous ne devriez donc pas rencontrer beaucoup de difficultés si vous désirez convertir des programmes d'un BASIC à l'autre. Il dispose également de quelques commandes supplémentaires dédiées au graphisme en couleur et aux possibilités sonores de l'Amstrad.

L'interpréteur BASIC est très important et occupe les 16K de la ROM supérieure. Ses capacités sont équivalentes de celles de la plupart des systèmes du commerce ; aucune possibilité n'a été réduite sous prétexte qu'il s'agit surtout d'un ordinateur familial. Il possède tout un éventail de fonctions consacrées aux chaînes (dont certaines sont originales, comme UPPER$ et LOWER$), toutes les fonctions numériques essentielles, il peut traiter des entiers ou des nombres réels, des tableaux multidimensionnels, etc. Il semble également bien adapté à l'écriture de programmes sérieux, de jeux d'arcade, de réalisation de graphes, et peut encore être fort utile pour les "devoirs de maths".

De plus, l'utilisation du clavier est extrêmement souple et l'ensemble du prochain chapitre sera consacré au clavier et au jeu de caractères.

## PROGRAMMATION STRUCTURÉE

Ce sujet a été largement exploré au cours des dernières années. Le BASIC de l'Amstrad offre les habituelles instructions de contrôle FOR...NEXT, GOSUB...RETURN, ON X GOSUB, ainsi que des instructions moins communes (mais très utiles) comme WHILE...WEND. Il permet à l'utilisateur de définir des fonctions — uniquement des fonctions ''à ligne unique'' — mais n'autorise ni les procédures ni les variables locales.

Bien que l'on puisse regretter l'absence de ces structures, on peut dire néanmoins qu'elles sont largement compensées par les possibilités de traitement de tâches multiples, qui facilitent de nombreuses structures difficiles à programmer de manière élégante avec des BASIC moins élaborés. Nous parlerons ultérieurement des tâches multiples dans ce chapitre.

## FICHIERS SUR CASSETTE

La gestion des entrées-sorties constitue une des tâches importantes que doit pouvoir faire n'importe quel BASIC. Il sera intéressant de voir jusqu'à quel point le BASIC de l'Amstrad sera adapté au lecteur de disquettes lorsque celui-ci pourra être connecté à l'unité centrale. Pour l'instant, bien sûr, nous devons nous contenter du lecteur de cassettes. Évidemment, un fichier sur cassette est toujours séquentiel, et il est facile à créer. Le programme suivant écrit dix nombres dans un fichier sur cassette intitulé NUMFICHE et les relit ensuite (rien que pour le plaisir !) :

```
 10 OPENOUT "NUMFICHE"
 20 FOR J = 1 TO 10
 30 WRITE #9, J
 40 NEXT J
 50 CLOSEOUT
 60 PRINT "RELECTURE DES DONNÉES"
 70 OPENIN "NUMFICHE"
 80 IF EOF THEN 150
 90 INPUT #9, J
100 PRINT J
110 GOTO 80
150 CLOSEIN
160 END
```

Ce programme est peut-être assez simple, mais il est intéressant d'observer certains points :

- Les noms des fichiers peuvent avoir une longueur maximale de 16 caractères. (Les noms plus longs sont réduits à 16 caractères avant d'être utilisés.)
- Pour placer des données dans un fichier, on emploie ordinairement l'instruction WRITE, mais l'on pourrait également utiliser PRINT. On préfère généralement WRITE, car cette instruction écrit les données d'une manière qui permet à INPUT de les relire directement.
- Le "canal" associé aux entrées-sorties du lecteur de cassettes est le canal 9, c'est pourquoi nous avons écrit "WRITE #9". Il n'est pas nécessaire de l'utiliser comme une constante. Lors de la mise au point d'un programme, il peut être préférable d'écrire, par exemple :

```
 20 OUTFICH=3
 30 WINDOW#3,30,40,1,5
 40 OPENOUT "NUMFICHE"
 ...
100 WRITE #OUTFICH
 ...
150 CLOSEOUT
```

  De cette manière, nous verrons que les données écrites apparaissent dans une petite fenêtre située en haut à droite de l'écran. Si tout semble fonctionner correctement, on peut modifier la ligne 20 en "OUTFICH=9" et supprimer la ligne 30.

Dans l'exemple précédent, les lignes 40 et 150 ne sont utiles que si le canal 9 est précisé à la ligne 20, mais elles ne gênent pas l'exécution du programme. Cependant, OPENOUT (ou OPENIN) affecte une zone tampon (un *buffer*) de 4K ; nous étudierons cela en détail plus tard.

Il peut être rassurant de voir s'afficher des messages sur l'écran pendant la lecture ou l'écriture de blocs de données sur une cassette, mais cela peut parfois être un inconvénient. Ces messages peuvent

être supprimés si l'on modifie le nom du fichier dans une instruction OPENxx. Voici un exemple :

```
OPENOUT "!NUMFICHE"
```

Le point d'exclamation n'est pas compris dans le nom du fichier, mais signifie simplement au système d'exploitation qu'il ne doit pas afficher sur l'écran des messages concernant la cassette.

Au premier abord, vous pourriez peut-être croire qu'il n'y a pas de commande de vérification ; certains systèmes en proposent ; vous pouvez ainsi vous assurer que la cassette est correctement écrite pendant que vous avez encore la possibilité d'effectuer une nouvelle copie des données ou du programme !

Il est vrai que l'Amstrad n'a pas de commande de vérification explicite, mais il nous offre la commande CAT, qui lit toute une cassette et affiche les noms des programmes ou des fichiers qui s'y trouvent. Ce n'est pas la même chose, dans le sens où cette instruction ne compare pas les données de la bande magnétique avec celles qui se trouvent en mémoire, mais ce qui importe est qu'elle puisse vérifier que la cassette est correctement lisible.

## PROGRAMMES SUR CASSETTE

Bien entendu, vous ne stockerez pas uniquement des données sur la bande magnétique, mais aussi des programmes. Un simple :

```
SAVE "PROG1"
```

pourra effectuer la sauvegarde de votre chef-d'œuvre. Il existe plusieurs manières de le recharger le lendemain. En voici quelques-unes :

- **LOAD "PROG1"** évitera tous les autres fichiers, s'il y en a, et chargera PROG1.
- **RUN "PROG1"** S'il est trouvé, le programme sera chargé et sera exécuté sans qu'aucune autre intervention ne soit nécessaire.
- **MERGE "PROG1"** effectuera une fusion de PROG1 avec le programme qui se trouve en mémoire, s'il y en a un, et remplacera les lignes dont le numéro est identique. Cette instruction est utile pour créer une bibliothèque de sous-programmes.

- **CHAIN "PROG1"** Le programme en cours sera effacé de la mémoire et PROG1 sera chargé, puis exécuté, mais les variables du programme en cours seront conservées et pourront ainsi être utilisées par PROG1.
- **CHAIN MERGE "PROG1"** fonctionne comme CHAIN, mais le programme existant sera conservé intégralement ou en partie (selon les numéros de ligne).
- N'oubliez pas que vous pouvez toujours utiliser les touches CTRL et ENTER pour éviter de taper RUN "".

En résumé, toutes les possibilités de liaisons de programmes que nous pouvons souhaiter sont à notre disposition.

Il n'y a pas que les programmes en BASIC qui peuvent être chargés (avec LOAD) ou sauvegardés (avec SAVE). Des sous-programmes en assembleur peuvent être chargés dans des zones de la mémoire, supérieures à celle occupée par le BASIC. SAVE et LOAD sont utilisés avec des paramètres supplémentaires qui précisent les positions de départ et de fin du sous-programme (de la "routine"). Les mêmes commandes peuvent être employées pour manipuler des tableaux situés en mémoire.

Les sous-programmes en assembleur seront traités dans la quatrième partie.

## L'ORGANISATION DE LA MÉMOIRE

Nous avons indiqué que le BASIC affecte 4K au tampon d'un fichier sur cassette, et qu'une partie de la mémoire peut être occupée par des sous-programmes. Ces deux exigences entraînent une réduction de la place disponible en BASIC, et la mémoire est utilisée en partant de la zone supérieure de cette limite ; dans un cas, de manière automatique (lorsque l'on emploie OPENIN ou OPENOUT), dans l'autre cas, grâce à la commande MEMORY qui effectue un "ajustement" de la limite supérieure.

Lorsque vous mettez le système en marche, l'espace utilisé par l'interpréteur BASIC s'étend de l'adresse (en code hexadécimal) &40 à l'adresse &AB7F, et l'instruction HIMEM, qui donne l'adresse de l'octet le plus élevé de la mémoire utilisée par le BASIC, possède la valeur &AB7F (ou 43903 en décimal, mais lorsqu'on parle d'adresse

la notation hexadécimale est mieux adaptée). Si vous souhaitez laisser une zone libre — au-dessus de celle utilisée par le BASIC — pour une routine en assembleur ou pour toute autre raison, vous devez employer la commande MEMORY afin de préserver cette zone. Supposons que vous chargiez un sous-programme (une routine) long de 128 octets, qui a été conçu pour être placé à l'adresse &AB00.

```
10 MEMORY &AAFF        REM LIMITER LA ZONE DU BASIC
                       SOUS LA ZONE DE LA ROUTINE
20 LOAD "!SUB1"        REM CHARGER LA ROUTINE
30 SUB1=&AB00          REM ADRESSE DE LA ROUTINE
...
100 CALL SUB1
```

Remarquez que la commande LOAD n'affecte pas le programme en BASIC qui se trouve déjà en mémoire, car la routine a été sauvegardée sous forme de fichier binaire. En fait, la sauvegarde aurait dû être accomplie par une instruction de ce genre :

```
SAVE "SUB1",B,&AB00,&B0,&AB00
```

qui permet d'enregistrer sur la cassette l'adresse de départ, la longueur et le point d'entrée, de sorte que LOAD puisse ultérieurement placer le fichier à la bonne adresse.

Si vous n'avez besoin d'utiliser la routine SUB1 que d'une manière temporaire, vous pourrez par la suite regagner de la place en récupérant la zone réservée. Si vous ajoutez au programme des instructions telles que :

```
5 ORIGMEMORY=HIMEM
```

et

```
200 MEMORY ORIGMEMORY
```

vous restaurerez la mémoire sous sa forme initiale. (Mais n'essayez plus d'appeler la routine SUB1!)

Ce qui précède a montré comment le sommet de la zone du BASIC peut être manipulé par le programmeur. Si le système a besoin de

4K de mémoire parce que vous avez ouvert un fichier sur cassette, il réduira automatiquement la valeur de HIMEM, comme vous pourrez vous en apercevoir si vous entrez :

```
PRINT HIMEM
OPENOUT "TEST"
PRINT HIMEM
```

et vous restituera, s'il le peut, la zone nécessaire au tampon dès qu'il aura fini de charger la routine :

```
CLOSEOUT
PRINT HIMEM
```

Toutefois, si vous réduisez HIMEM après que les 4K auront été affectés par le système, celui-ci ne pourra pas "remonter" HIMEM après avoir rencontré l'instruction CLOSEOUT, et il lui laissera sa valeur !

```
OPENOUT "TEST"
PRINT HIMEM
MEMORY HIMEM-1
CLOSEOUT
PRINT HIMEM
```

Les 4K ne seront quand même pas perdus, car si vous ouvrez par la suite un fichier sur cassette, le BASIC utilisera la même zone tampon au lieu d'en occuper une nouvelle. Malgré tout, vous préférerez peut-être définir la zone réservée à votre routine avant d'ouvrir le moindre fichier sur cassette, et de cette manière le tampon du lecteur de cassette sera libéré après la fermeture du fichier.

L'utilisation de l'instruction MEMORY ou celle du lecteur de cassette ne sont pas les seules raisons pour lesquelles HIMEM est modifiée, comme nous le verrons dans le prochain chapitre, qui explique également comment sont employés les 128 octets situés entre &AB00 et &ABFF.

## LE TRAITEMENT DES TÂCHES MULTIPLES

Nous avons gardé pour la fin du chapitre l'une des caractéristiques les plus intéressantes de ce BASIC : le traitement des tâches multiples.

Normalement, un programme n'exécute qu'une tâche à la fois. Lorsque vous tapez RUN, le programme commence par la première instruction et se poursuit jusqu'au moment où il rencontrera l'instruction END. Évidemment, la plupart des programmes contiennent des boucles, car les ordinateurs exécutent les instructions à une très grande vitesse ; sans cela les temps d'exécution, même ceux des programmes très longs, seraient extrêmement courts.

On s'attend parfois à ce que le programme fasse une pause — par exemple, afin de recevoir des informations du clavier — mais on sait qu'il parcourt les instructions d'une manière prévisible et séquentielle.

Bien entendu, le système interrompt de temps en temps le déroulement du programme. Nous savons, par exemple, qu'une interruption se produit toutes les 20 millisecondes pour permettre un rafraîchissement de l'écran, de manière à conserver un affichage constant. Le système doit également accomplir d'autres corvées ménagères, comme l'avance du compteur (TIME) qui fonctionne comme une horloge du système.

Le BASIC de l'Amstrad permet à l'utilisateur de demander que son programme soit interrompu à des intervalles de temps de son choix afin d'effectuer des tâches prioritaires. Chaque tâche a un niveau de priorité, et une tâche de priorité inférieure peut être interrompue par une tâche ayant une priorité supérieure. Voici un exemple de programme :

```
  10 EVERY 100,0 GOSUB 3000
  10 CLS
  30 ...
 999 END
3000 REM TACHE DE PRIORITE 0
3010 ...
3999 RETURN
```

Le programme principal va de l'instruction 10 à l'instruction 999. L'instruction 10 entraîne la création d'une sous-tâche ayant un niveau de priorité de 0. Cette tâche (ou sous-tâche) pourra exécuter toutes les instructions nécessaires, mais devra tôt ou tard atteindre l'instruction 3999.

Après l'instruction 10, le programme principal ne reviendra sans doute jamais à cette sous-tâche. Il continuera d'exécuter ses propres instructions, sachant que le système d'exploitation a entamé un

compte à rebours et que la sous-tâche sera exécutée toutes les 2 secondes (le compte se fait en cinquantièmes de seconde et nous en avons demandé 100 à la ligne 10).

Quand la sous-tâche (le sous-programme) reçoit le contrôle de l'exécution à la ligne 3000, elle sait qu'elle possède un contrôle absolu sur le déroulement des opérations jusqu'au moment où elle rencontrera l'instruction 3999 et rendra le contrôle au programme principal qui poursuivra son travail interrompu. Les sous-tâches ont une priorité supérieure à celle du programme principal.

Il s'agit d'une caractéristique particulièrement utile pour toutes sortes d'applications. Dans l'exécution d'un jeu, cela peut servir à déplacer des figures sur l'écran à intervalles réguliers (voir les jeux dans la troisième partie), ou à afficher et faire avancer une horloge sur l'écran :

```
   2 ON BREAK GOSUB 3000
   3 CLS
   5 WINDOW #0,1,30,1,25 : WINDOW #1,31,40,25,25
  10 INPUT "Donnez les heures, les minutes, les secondes: ",
     HEURES,MINS,SECS
  20 EVERY 50 GOSUB 2000
  30 FOR J=1 TO 500
  40 PRINT J
  50 NEXT J
  60 GOTO 30
2000 REM TACHE DE PRIORITE 0 TOUTES LES SECONDES
2005 SECS=SECS+1
2010 IF SECS=60 THEN SECS=0:MINS=MINS+1
2020 IF MINS=60 THEN MINS=0:HEURES=HEURS+1
2030 IF HEURES=25 THEN HEURES=0
2100 PRINT #3, USING "##:##:##";HEURES,MINS,SECS
2999 RETURN
3000 REM ARRET
3010 MODE 1
3020 STOP
```

Le programme précédent nous montre une tâche principale assez ennuyeuse allant de la ligne 5 à la ligne 60, une sous-tâche aux lignes 2000-2999 et un sous-programme, à la ligne 3000, pour arrêter l'exécution du programme si vous en avez assez.

Bien entendu, vous pouvez penser qu'une interruption toutes les secondes constitue une servitude inutile. Dans ce cas, modifiez la ligne 20 en EVERY 3000 (au lieu de EVERY 50), modifiez la ligne 10 pour n'entrer que les heures et les minutes, et supprimez l'affichage des secondes dans l'exécution de la sous-tâche (ligne 2100). Ainsi, votre horloge n'avancera que toutes les minutes.

Il est deux points importants dont il faut tenir compte lorsqu'on construit ce genre de sous-programme :

- Assurez-vous que le programme principal et le sous-programme n'utilisent pas involontairement les mêmes variables. Par exemple, si le sous-programme modifiait la variable J, le programme principal pourrait donner des résultats imprévisibles !
- Si vous exécutez le programme précédent, puis pressez une fois sur la touche ESC, le programme s'arrête provisoirement (c'est une pause). Si vous pressez ESC une seconde fois, le programme s'arrête définitivement (achèvement de l'exécution). Mais si vous pressez la barre d'espacement, le programme continue. Faites cela après dix secondes de pause, et regardez avec quelle rapidité l'horloge se remet à l'heure ! Cela nous montre que le système n'a pas oublié les interruptions qu'il n'a pas pu exécuter, mais qu'elles sont enregistrées et qu'il les exécute plus tard, dès qu'il le peut (jusqu'à un maximum d'environ 127 interruptions).

Ainsi, l'exécution de sous-tâches peut être assez aisée, mais il y a quelques précautions à prendre. Dans l'exemple précédent, la sous-tâche affichait l'heure dans une fenêtre *(window)* différente de celle utilisée par le programme principal. Que se passerait-il si le sous-programme et le programme principal utilisaient la même fenêtre ? Envisageons le cas d'un programme qui possède le sous-programme suivant :

```
100 LOCATE X,Y
104 PEN 2
106 PRINT A,B,C
110 FOR J=1 TO 25
120 ...
```

et une sous-tâche contenant cette instruction :

```
4020 LOCATE XPLACE,YPLACE : PEN3 : PRINT A$
```

Si l'interruption de la sous-tâche se produit entre les instructions 106 et 110, ou entre 110 et 120, tout peut aller pour le mieux, mais vous verrez ce qui se passe si l'interruption a lieu quand le programme se trouve entre les lignes 100 et 104 — nous nous sommes soigneusement placés en X,Y et, avant d'avoir la moindre chance d'afficher quoi que ce soit, nous sommes entraînés à la position XPLACE,YPLACE. Lorsque l'exécution de la sous-tâche se termine (par une instruction RETURN), le programme principal passe en PEN 2 et affiche A, B et C — mais bien entendu il y a très peu de chances pour que la position soit alors correcte et le résultat paraîtra plutôt bizarre.

Le problème est encore plus grave si l'interruption se produit après l'exécution de l'instruction 104 ; non seulement les valeurs qui s'afficheront ensuite seront mal placées, mais leur couleur sera incorrecte !

Fort heureusement, ce problème peut facilement être évité si on a bien compris pourquoi il risque de se poser. Le BASIC de l'Amstrad nous offre les instructions DI (*Disable Interrupt,* annulation d'une interruption) et EI (*Enable Interrupt,* rétablissement d'une interruption) ; nous pouvons les placer au début et à la fin des passages délicats du programme. Dans l'exemple précédent :

```
99 DI
108 EI
```

pourraient résoudre notre problème, en nous assurant que nous ne risquons pas d'être interrompus entre le positionnement (ligne 100) et l'affichage (ligne 106). Naturellement, nous devons limiter autant que possible le nombre des instructions situées entre DI et EI, car nous ne souhaitons pas que les interruptions demeurent trop longtemps dans la ''file d'attente'' ; cependant, nous savons qu'elles ne seront pas oubliées, mais simplement retardées, et que rien ne sera perdu.

Ne confondons pas les ''interruptions'' dont nous parlons ici avec les interruptions du système qui se produisent au moins toutes les 20 millisecondes. DI n'a aucun effet sur les interruptions propres au système.

L'exemple précédent nous a montré une sous-tâche, mais on peut en avoir jusqu'à 4 au maximum, ce qui fait un total de 5 si l'on compte le programme principal. Le numéro de la sous-tâche, de 0 à 3, est indiqué par le second paramètre de l'instruction EVERY, comme ceci :

```
10 EVERY 100,1 GOSUB 1500
20 EVERY 80,2 GOSUB 3400
```

Ces deux lignes introduisent deux sous-tâches, numérotées 1 et 2. Comme d'habitude, 0 est le numéro de tâche "par défaut" (implicite).

Il s'agit ici d'une priorité stricte ; la sous-tâche n° 3 ayant la priorité maximale et le programme principal ayant la priorité minimale. Par conséquent, si une sous-tâche de priorité 1 est en cours d'exécution, elle peut être interrompue par une sous-tâche de priorité 2 ou 3, mais les interruptions associées à une sous-tâche de priorité 0 seront "retardées" jusqu'à la fin de l'exécution de la sous-tâche 1, qui passera alors le contrôle à la sous-tâche 0 au lieu de le rendre au programme principal.

Jusqu'à présent, nous avons simplement cité la commande EVERY, qui permet des interruptions régulières. Il existe également la commande AFTER, qui ne produit qu'une interruption.

Bien entendu, une sous-tâche peut lancer une autre sous-tâche, ou se relancer elle-même. Nous pouvons avoir :

```
  10 AFTER 1000 GOSUB 3000
  ....
3000 REM SOUS-TACHE 0
  ....
3100 AFTER 500 GOSUB 3000
3110 RETURN
```

La fonction REMAIN nous est également très utile pour empêcher des interruptions ultérieures. REMAIN (numéro de sous-tâche) annule toutes les interruptions en attente associées à la sous-tâche en question. Dans tous les cas, les sous-tâches sont annulées lorsque le programme rencontre une commande END.

## QUELQUES MOTS D'AVERTISSEMENT

Vous pouvez vous rendre compte que la création des sous-tâches vous est particulièrement facilitée. Les quelques rares pièges possibles peuvent être aisément évités.

### Piège n° 1

Nous en avons déjà parlé. Vérifiez que vous n'employez pas involontairement dans une sous-tâche une variable qui est également utilisée dans un autre sous-programme ou dans le programme principal, car vous risqueriez d'obtenir des résultats imprévisibles.

### Piège n° 2

Assurez-vous de placer entre DI et EI les parties de programme qui ne doivent pas être interrompues.

### Piège n° 3

Il est inutile d'écrire EVERY 50 GOSUB 3000 s'il faut plus d'une seconde pour exécuter la sous-tâche qui débute en 3000 ! Vous resterez tout le temps dans la sous-tâche et le programme principal n'aura jamais l'occasion de reprendre le contrôle... Si cela s'avère nécessaire, vous pouvez vérifier la durée d'exécution d'un sous-programme en plaçant au début l'instruction T=TIME, et à la fin l'instruction PRINT TIME-T. La valeur affichée donnera la durée (l'unité est le 1/300 de seconde).

En résumé, les commandes d'interruption et de traitement de tâches multiples de l'Amstrad sont remarquablement utiles et puissantes ; elles permettent de traiter facilement des situations qui seraient complexes avec d'autres BASIC.

## EXTENSIONS FUTURES

Un excellent présage pour l'avenir est que le BASIC est conçu pour être étendu, et les extensions futures ne devraient pas être des bricolages à retardement (comme c'est manifestement le cas dans bien d'autres systèmes !). La syntaxe est la suivante : les commandes précédées d'une barre verticale "|" seront considérées par le BASIC comme des commandes externes, c'est-à-dire des commandes qui seront traitées par une autre ROM.

Reste à savoir comment cela fonctionnera ; mais le contrôleur de disquette de l'Amstrad contiendra sans doute une ROM qui comportera quelques nouvelles commandes en BASIC afin de permettre l'affichage des catalogues de disquettes, la lecture et l'écriture de fichiers

à accès direct, etc. Grâce à cette convention concernant les ''commandes externes'', la nouvelle ROM pourra effectivement étendre les capacités de la ROM actuelle, en remplaçant des fonctions existantes ou en rajoutant de nouvelles fonctions.

Ce n'est encore qu'une simple conjecture, mais ce serait une méthode beaucoup moins coûteuse et moins douloureuse que la méthode (classique !) consistant à retirer les anciennes ROM pour les remplacer par des nouvelles !

# 3

# TOUCHES ET CARACTÈRES

Le clavier de l'Amstrad est d'une conception très satisfaisante, à la fois par son confort d'utilisation et grâce au logiciel qui lui est associé. Dans ce chapitre, nous examinerons la police de caractères de l'Amstrad, les moyens de la modifier et la manière dont le clavier peut être adapté à vos besoins.

## CARACTÈRES DÉFINIS PAR L'UTILISATEUR

Au cours du Chapitre 2, nous avons vu comment la limite supérieure du BASIC pouvait être "remontée" ou "abaissée". Il existe une autre fonction du BASIC qui provoque une réduction de la valeur de HIMEM : c'est la commande SYMBOL AFTER. L'Amstrad autorise l'affichage de 256 caractères différents. Normalement, les 32 premiers caractères ne sont pas visibles, car ils sont interprétés comme des caractères de contrôle, mais en fait vous pouvez les afficher en les faisant précéder de CHR$(1).

```
10 REM AFFICHAGE DE TOUS LES CARACTERES, 0 A 255
20 FOR J=0 TO 31
30 PRINT CHR$(1);
40 PRINT CHR$(J);
50 NEXT J
60 FOR J=32 TO 255
70 PRINT CHR$(J);
80 NEXT J
90 END
```

Une caractéristique très utile de l'Amstrad permet à n'importe lequel de ces 256 caractères d'être remplacé par un autre, et c'est pourquoi vous pouvez concevoir des caractères spéciaux qui vous conviennent. Quand vous mettez en marche le système, vous disposez

déjà de la possibilité de redéfinir jusqu'à 16 caractères (les numéros 240 à 256), mais vous pouvez améliorer cette possibilité afin de pouvoir redéfinir les 256 caractères si vous le désirez.

Comme les définitions des caractères sont normalement contenues en ROM (sauf celles des 16 derniers caractères) et que vos propres définitions devront évidemment se trouver en RAM, le système doit disposer d'une zone réservée ; c'est pourquoi la valeur de HIMEM est réduite si vous entrez la commande SYMBOL AFTER. L'instruction SYMBOL AFTER 200 signifie que vous souhaitez pouvoir redéfinir tous les caractères à partir du 200$^{e}$. SYMBOL AFTER 0 permet de redéfinir les 256 caractères.

Ayant utilisé SYMBOL AFTER, si cela s'avère nécessaire (n'oubliez pas qu'il existe implicitement un SYMBOL AFTER 240 lorsque vous déclenchez le système) vous pouvez utiliser la commande SYMBOL pour définir le nouveau caractère. Chaque caractère est considéré comme une matrice de 8 × 8 pixels ; vous placez un pixel à l'état "ON" (affichage) en lui donnant la valeur 1 ou vous le placez à l'état "OFF" (non-affichage) en lui donnant la valeur 0, en commençant par la première rangée de la matrice. Le caractère A — CHR$(65) —, par exemple, pourrait être défini comme ceci :

```
SYMBOL 65, &18,&3C,&66,&66,&7E,&66,&66,0

   ...**...  &18
   ..****..  &3C
   .**..**.  &66
   .**..**.  &66
   .******.  &7E
   .**..**.  &66
   .**..**.  &66
   ........  &00
```

Ne nous imaginons pas qu'un pixel est identique à un simple point de l'écran. Cela est exact en mode 2 (haute résolution) ; mais en mode 1, un pixel est représenté par deux points, et en mode 0 par quatre points.

Remarquons que si nous entrons une seconde commande SYMBOL AFTER, tous les caractères redéfinis précédemment seront effacés ! De plus, si vous avez ajusté la limite supérieure de la zone utilisée par le BASIC depuis votre dernière commande SYMBOL AFTER, le

système refusera la nouvelle commande SYMBOL AFTER. Cela explique pourquoi vous ne pouvez pas employer MEMORY afin de préserver une zone pour un sous-programme, et lancer ensuite une instruction SYMBOL AFTER. Le système a exécuté sa commande implicite SYMBOL AFTER 240 lorsqu'il a déclenché le BASIC, et vous déplacez ensuite la limite supérieure grâce à une commande MEMORY. Le remède consiste à lancer l'instruction SYMBOL AFTER avant toute commande MEMORY ou avant d'utiliser un fichier sur cassette.

En fait, c'est dans les 128 octets commençant à l'adresse &AB80 que le système stocke les définitions des caractères 240 à 255, à raison de 8 octets par caractère.

## ÉTUDE DE CARACTÈRES

Le programme qui suit peut vous servir à tester vos propres caractères. Il utilise le MODE 0 pour plus de clarté. Employez les touches flèches pour vous déplacer dans la matrice. En pressant la touche A, quelle que soit votre position, vous mettez le pixel associé au curseur à l'état "ON" (allumé), en pressant E vous éteignez le pixel ("non-affichage"), et en pressant Q vous arrêtez le programme et faites afficher la définition utilisée par la commande SYMBOL. Le caractère est affiché au format réel à mesure qu'il est défini, et le code hexadécimal de chaque rangée de pixels est affiché pour vous permettre de mieux saisir comment le caractère est défini dans une commande SYMBOL.

```
1 REM DEFINITIONS DES CARACTERES PAR L'UTILISATEUR
5 ON BREAK GOSUB 6000
10 GOSUB 1000 : REM initialisation
20 GOSUB 2000 : REM conception des caractères
30 END
1000 MODE 0
1010 LOCATE 1,1
1020 FOR RAN=1 TO 8
1030  FOR COL=1 TO 8
1040   PRINT ". ";
1050  NEXT COL
```

```
1060  PRINT "&O"
1080  PRINT
1090 NEXT RAN
1100 LOCATE 5,22 : PRINT "Tapez A (Allumer), E (Eteindre),
     Q (Quitter) ou pressez une touche fléchée."
1200 WINDOW #1,10,10,20,20
1210 PEN #1,2
1220 SYMBOL AFTER 128
1230 DIM PRAN(8,8),PXRAN(8)
1240 GOSUB 3000 : REM DEFINIR CARACTERE 128
1250 DOT$=CHR$(143)
1260 UNDOT$="."
1270 DOTCURSOR$=CHR$(127)
1280 UNDOTCURSOR$="*"
1290 LOCATE 1,1 : X=1 : Y=1 : PRINT UNDOTCURSOR$;
1999 RETURN
2000 REM
2010 Z$=INKEY$ : IF Z$="" THEN 2010
2015 Z$=UPPER$(Z$)
2020 Z=INSTR("AEQ"+CHR$(240)+CHR$(241)+CHR$(242)+C
     HR$(243),Z$)
2030 IF Z=0 THEN 2010
2035 NX=0 : NY=0
2040 ON Z GOSUB 2100,2200,2300,2400,2500,2600,2700
2045 LOCATE X*2-1,Y*2-1
2050 IF PRAN(X,Y)=1 THEN PRINT DOT$; ELSE PRINT UNDOT$;
2060 X=X+NX : Y=Y+NY : LOCATE X*2-1,Y*2-1
2070 IF PRAN(X,Y)=1 THEN PRINT DOTCURSOR$; ELSE
     PRINT UNDOTCURSOR$;
2080 LOCATE 19,Y*2-1 : PRINT "  " : LOCATE 19,Y*2-1 :
     PRINT HEX$(PXRAN(Y))
```

```
2090 GOSUB 3000
2095 PRINT #1,CHR$(128);
2099 GOTO 2010
2100 REM A (ALLUMER)
2110 PRAN(X,Y)=1
2120 PXRAN(Y)=PXRAN(Y) OR 2^(8-X)
2199 RETURN
2200 REM E (ETEINDRE)
2210 PRAN(X,Y)=0
2220 PXRAN(Y)=PXRAN(Y) XOR 2^(8-X)
2299 RETURN
2300 REM Q (QUITTER)
2310 LOCATE 1,22 : PRINT CHR$(20);:PRINT
     "CARACTERE X,";
2320 FOR RAN=1 TO 7
2330 PRINT "&";HEX$(PXRAN(RAN));",";
2340 NEXT RAN
2350 PRINT "&";HEX$(PXRAN(RAN));
2360 END
2400 REM FLECHE HAUT
2410 IF Y>1 THEN NY=1
2499 RETURN
2500 REM FLECHE BAS
2510 IF Y<8 THEN NY=1
2599 RETURN
2600 REM FLECHE GAUCHE
2610 IF X>1 THEN NX=-1
2699 RETURN
2700 REM FLECHE DROITE
2710 IF X<8 THEN NX=-1
2799 RETURN
```

```
3000 REM DEFINITION CARACTERE 128
3010 SYMBOL 128,PXRAN(1),PXRAN(2),PXRAN(3),PXRAN(
        4),PXRAN(5),PXRAN(6),PXRAN(7),PXRAN(8)
3999 RETURN
6000 REM ARRET
6010 MODE 1
6020 STOP
```

## CARACTÈRES SUR LA TOUCHE

Si vous créez vos propres caractères en employant la méthode décrite précédemment, vous voudrez peut-être pouvoir les afficher sur l'écran en frappant simplement une touche plutôt qu'en entrant, par exemple, PRINT CHR$(220). Cela peut se faire assez facilement car l'Amstrad vous permet de redéfinir n'importe quelle touche en lui donnant n'importe quelle valeur — ou plus exactement n'importe laquelle parmi 3 valeurs, car la touche peut être "normale", ou bien utilisée en pressant simultanément la touche CTRL ou la touche SHIFT.

De plus, vous pouvez choisir ou non la répétition automatique d'une touche, c'est-à-dire la répétition de l'affichage d'un caractère tant que la touche est maintenue enfoncée. Vous pouvez même tester la vitesse de répétition pour trouver une vitesse qui vous convient, grâce à la commande SPEED KEY.

Pour comprendre cela, vous devez savoir que chaque touche possède un numéro. Vous devez penser que la deuxième touche de la troisième rangée, en partant de la gauche, est la "touche A", mais c'est uniquement par convention qu'elle affiche la lettre A, et elle peut être modifiée, si vous le désirez, pour vous fournir un autre caractère. Mais le numéro de la touche ne change jamais — ou plutôt devrions-nous dire le numéro de la touche ou du *joystick* (manette de jeu) car les manettes de jeu sont considérées comme faisant partie du clavier.

Un tableau présentant les numéros des clefs est contenu dans le manuel de l'Amstrad, Appendice III, page A3.16. En se référant au numéro de la touche, nous pouvons modifier le caractère associé à

une touche quelconque, soit lors d'une frappe simple, soit lorsqu'elle est frappée conjointement avec une pression sur la touche SHIFT ou la touche CTRL. Par exemple :

```
KEY DEF 69,1,&62,&42,2
```

indique au système de modifier la touche 69 (qui donne normalement a, A, ou CTRL–A) pour faire en sorte qu'elle donne b, B ou CTRL–B. La valeur 1 signifie que nous souhaitons une répétition du caractère tant que la touche est enfoncée. Faites un essai !

L'ensemble du clavier peut être reprogrammé de cette manière si vous le désirez (sauf les touches SHIFT et CTRL), et vous pouvez ainsi transformer le clavier QWERTY en un clavier DVORAK, ou en un clavier AZERTY (sans pouvoir toutefois changer les caractères disposés sur les touches elles-mêmes, malheureusement).

Transformer la touche A pour qu'elle donne la lettre B ne paraît pas très sensé, mais vous pouvez aussi bien la modifier pour qu'elle vous donne un caractère peu usité, que vous aurez défini vous-même. Supposons que vous ayez utilisé la commande SYMBOL pour créer 3 symboles graphiques, 220, 221 et 222. Vous pouvez peut-être également vous dispenser pendant un moment des touches de crochets.

En vous référant à l'Appendice III du manuel de l'Amstrad, vous verrez que le numéro de la touche du crochet gauche est le 17 et que celui du crochet droit est le 19. Vous pouvez alors entrer l'instruction suivante :

```
KEY DEF 17,1,220,221,222
```

qui modifiera la touche du crochet gauche afin qu'elle affiche le symbole 220 en mode "normal", le caractère 221 en mode majuscule (SHIFT) et le symbole 222 si on frappe la touche en pressant simultanément CTRL. (La valeur 1 signifie que la répétition est validée.) La touche 19 peut être préservée pour d'autres usages.

En examinant le clavier, nous verrons qu'il y a de nombreuses touches "de réserve" qui peuvent être employées de cette manière.

## DES PHRASES ENTIÈRES SUR UNE TOUCHE

On pourrait penser que l'exemple précédent à démontré la grande flexibilité du clavier de l'Amstrad. Mais ce n'est pas tout ! Vous pou-

vez redéfinir n'importe quelle touche pour que, lorsqu'on la presse, elle affiche une phrase entière plutôt qu'un seul caractère.

Cela est possible en réservant certaines valeurs (entre 128 et 140) comme étant des ''caractères d'expansion''. Lorsqu'une touche donne une de ces valeurs, le système consulte un ''tableau des phrases'' (il s'agit de chaînes de caractères) et en extrait le caractère ou la chaîne appropriée. Ainsi, le caractère d'expansion 128 peut faire afficher LIST, le caractère 129 peut faire afficher RENUM, etc.

En fait, le système prévoit, ''par défaut'', que les touches du pavé numérique donnent les valeurs 128 à 140, qui sont définies au départ pour afficher les valeurs ''0'' à ''9'' du code ASCII, plus le point décimal. De plus, la petite touche ENTER du pavé numérique a été définie de telle sorte que, si on la presse simultanément avec la touche CTRL, la chaîne RUN ''+CHR$(13) soit affichée. Cela explique pourquoi, dans ce cas, il faut utiliser la petite touche ENTER, et non la grande.

Cet emploi des caractères d'expansion et l'état initial du ''tableau de chaînes'' sont présentés dans le manuel, Appendice III, page 13.15.

En tapant la commande KEY (qu'il ne faut pas confondre avec la commande KEY DEF employée précédemment), vous pouvez établir une liste de chaînes dont chacune sera associée à un caractère d'expansion particulier. Par exemple, si vous écrivez un long programme qui utilise différentes couleurs, encres, etc., vous pourriez apprécier de disposer d'une chaîne comme celle-ci :

```
CLS:INK 0,1:INK 1,24:PAPER 0:PEN 1:LIST
```

qui redonnerait à l'écran une présentation convenable. Mais cette instruction est assez longue à entrer, surtout si vous écrivez en vert vif sur un fond vert citron !

Malheureusement, vous ne pouvez pas assigner toute la chaîne à une seule touche, car la limite est de 32 caractères par chaîne. Mais vous pouvez la couper en deux parties.

```
KEY 138,CHR$(13)+"CLS:INK 0,1:INK 1,24"+CHR$(13)
KEY 139,CHR$(13)+"PAPER 0:PEN1:LIST"+CHR$(13)
```

Ces deux lignes permettront à la touche du point décimal et à la petite touche ENTER d'afficher les instructions voulues quand vous le désirerez.

Tout cela fait montre d'une considérable ingéniosité. Par exemple, si vous définissez une touche pour qu'elle affiche LIST, vous pouvez, soit enfermer la chaîne entre deux CHR$(13) afin qu'une pression sur la touche puisse produire immédiatement un défilement du programme, soit ne pas inclure le second CHR$(13) de fin de ligne. Dans ce dernier cas, la commande LIST est affichée, mais le système attend que vous complétiez l'instruction. Vous pouvez alors choisir de presser la touche ENTER (la grande touche, pas la petite), ou de sélectionner les lignes du programme qui seront affichées.

Deux questions peuvent se poser à vous :

- Que se passe-t-il si vous souhaitez, par exemple, que la touche "7" du pavé numérique affiche le CHR$(135) au lieu de considérer 135 comme un caractère d'expansion à chercher dans le tableau des chaînes ? La réponse consiste simplement à insérer CHR$(135) dans le tableau des chaînes :

```
KEY 135,CHR$(135)
```

- Que se passe-t-il si vous lisez les informations en provenance du clavier en utilisant la fonction INKEY$ et qu'une des touches pressées donne une chaîne d'expansion ? La réponse est que la fonction INKEY$ prendra comme valeur le premier caractère de la chaîne ; les instructions INKEY$ suivantes vous donneront les autres caractères de la chaîne.

Cette caractéristique vous offre une remarquable souplesse, et elle manque à de nombreux systèmes plus gros et plus coûteux. Employée à bon escient, elle peut faire gagner beaucoup de temps. Les restrictions sont qu'aucune chaîne ne peut posséder plus de 32 caractères et que le total des caractères d'expansion ne peut pas être supérieur à 100 — vous ne pouvez donc pas définir les touches pour qu'elles affichent tous les mots clefs du BASIC.

N'oubliez pas que vous n'êtes pas tenu de vous limiter au pavé numérique. Chaque touche du clavier peut être définie pour donner une valeur comprise entre 128 et 159 grâce à la commande KEY DEF, et il vous est ensuite possible d'employer KEY pour assigner une chaîne à la touche ainsi définie.

N'oubliez pas non plus que chaque touche est capable de produire 3 caractères "normaux" ou 3 caractères d'expansion, puisqu'une tou-

che est définie pour une frappe "en minuscule", "en majuscule" (avec SHIFT) et conjointement avec CTRL.

C'est simple ! Mais un avertissement est quand même nécessaire : n'oubliez pas que la touche CAPS LOCK peut placer automatiquement une touche en majuscule et que, si vous avez défini l'état de cette touche en position majuscule en lui assignant une chaîne de caractères, vous risquez d'avoir des surprises... Et ne cherchez pas à modifier la touche ESC. Apparemment, cela dérange le BASIC et le système risquerait de se réinitialiser de manière imprévisible.

## LE TAMPON ASSOCIÉ AU CLAVIER

Afin d'achever notre exploration du clavier, nous devons encore examiner le tampon qui lui est associé. Sur de nombreuses machines, vous risquez de perdre un caractère parce que le BASIC est trop lent pour votre vitesse de frappe. C'est affreusement ennuyeux. Mais toutes les 20 ms le BASIC de l'Amstrad effectue une vérification afin de savoir si une touche a été pressée, et dans l'affirmative il place le caractère correspondant dans un tampon. Lorsque vous travaillez en utilisant INKEY$, le système n'attend pas immédiatement qu'une touche soit pressée, mais prend le premier caractère en attente dans le tampon, s'il s'en trouve un. Vous pouvez donc entrer vos caractères indépendamment de ce qu'accomplit le programme. Pour le vérifier, exécutez le programme suivant :

```
10 FOR J=1 TO 10
20 Z$=INKEY$:IF Z$="" THEN 20
30 FOR J1=1 TO 1000:NEXT J1
40 PRINT Z$;" ";
50 NEXT J
```

et observez comment le système progresse vaillamment, sans renverser une seule goutte...

# 4

# LA SPLENDEUR DU TECHNICOLOR

Dans ce chapitre, nous étudierons l'écran de l'Amstrad, comment cet écran est géré et comment s'effectue la sélection des couleurs. Cela nous donnera une base avant de passer à des possibilités graphiques plus étendues.

Toute sélection de couleur doit s'appuyer sur une compréhension claire des éléments que nous offre le BASIC. (Espérons que vous disposez d'un moniteur couleur.)

## LA MÉMOIRE ÉCRAN

L'écran de l'Amstrad peut parfois sembler bien paisible, mais en vérité il accomplit constamment une activité frénétique. Un moniteur ou un poste de TV ne retient pas longtemps une image ; il doit être "rafraîchi" à intervalles fréquents. Dans le cas de l'Amstrad, le système "repeint" l'ensemble de l'écran 50 fois par seconde, c'est-à-dire toutes les 20 ms. Comme l'œil humain ne peut pas suivre à une telle vitesse, l'image de l'écran lui apparaît fixe.

Les données ou les images affichées sur l'écran sont contenues dans une zone de la mémoire principale de l'ordinateur, débutant normalement à l'adresse &C000 (mais voyez le Chapitre 5). Si nous écrivons quelque chose dans cette zone, cela sera affiché sur l'écran en 20 ms, c'est-à-dire la prochaine fois que l'écran sera rafraîchi. L'affichage nous semblera donc instantané.

Malheureusement, vous ne parviendrez pas à écrire des données cohérentes sur l'écran en utilisant POKE pour les placer en mémoire à partir de l'adresse &C000. La raison en est que la topographie de la mémoire de l'écran est très compliquée ; un simple octet ne correspond pas à un caractère, et une instruction POKE &C000,65, par exemple, ne fera pas apparaître un A majuscule en haut à gauche de l'écran. Malheureusement !

La complexité de la zone mémoire associée à l'écran ne nous occupe généralement pas, car il y a des programmes, à la fois en BASIC et en assembleur, pour gérer l'écran à notre place. Si vous pensez qu'il vous faut absolument connaître la relation entre les pixels et les octets, vous devrez vous procurer le manuel technique de l'Amstrad.

## MODE D'AFFICHAGE

Nous avons le choix entre 3 types d'affichage. Il existe 3 modes d'écran utilisables, et à tout instant notre choix représentera une relation entre la couleur et la résolution. Chaque pixel ("la plus petite portion d'écran" adressable) doit être défini individuellement.

En mode 1, qui est le mode "par défaut" dans lequel se place le système lors de sa mise en marche, nous avons 25 rangées de 40 colonnes, autrement dit 1 000 caractères à définir en mémoire, et la zone associée à l'écran dans l'Amstrad est de 16K, allant généralement de l'adresse &C000 à l'adresse &FFFF. Il y a donc 16 octets par caractère, plus quelques octets laissés pour compte.

Comme nous l'avons vu dans le Chapitre 3 (caractères définis par l'utilisateur), un caractère consiste en fait en une matrice de 64 pixels, et il y a donc 16/64 octets par pixel, c'est-à-dire 2 bits. Avec 2 bits, on peut représenter 4 états : 00, 01, 10 ou 11 (autrement dit : 0, 1, 2 ou 3). On peut donc considérer chaque pixel comme ayant une couleur parmi quatre — mais pas plus.

Maintenant, vous devriez facilement comprendre pourquoi le mode 0 permet d'avoir 16 couleurs (parce qu'il y a 4 bits par pixel) et pourquoi le mode 2 n'en autorise que 2 (il n'y a qu'un seul bit par pixel !) Dans tous les modes, la zone mémoire allouée à l'écran est toujours de 16K.

## UN TEST SUR LES COULEURS

Donc, en revenant au mode par défaut, le mode 1, nous sommes constamment limités à 4 couleurs (bien que toutes les couleurs puissent être affichées par clignotement, en faisant alterner 2 couleurs, si vous tenez vraiment à finir par avoir mal au crâne !). Commençons par réinitialiser le système. Le signal READY s'affiche en jaune sur fond bleu — si vous possédez un moniteur couleur ! Manifestement, seules 2 des 4 couleurs disponibles sont mises en évidence pour l'instant.

Entrez le petit programme suivant :

```
  10 CLS
  20 GOSUB 1000
  30 END
1000 FOR J=1 TO 5
1010 PRINT J;
1020 NEXT J
1030 PRINT
1040 RETURN
```

Exécutez le programme. Nous comparons souvent l'écriture des caractères sur l'écran à l'utilisation d'un stylo sur du papier. Nous pouvons donc dire que le papier est bleu et que nous avons écrit avec un stylo jaune.

Entrez maintenant les modifications suivantes dans le programme précédent :

```
30 PEN 2
40 GOSUB 1000
50 PEN 3
60 GOSUB 1000
70 END
```

et vous devriez voir 3 séries de caractères sur l'écran, chacune étant d'une couleur différente. Avec le fond bleu, cela nous donne bien 4 couleurs. (Il est exact que nous pourrions colorer la bordure avec une cinquième couleur permanente, ou avec une cinquième et une sixième couleurs qui apparaîtraient alternativement ; cependant, nous ne pouvons rien faire de la bordure, à part la colorer, et elle n'est pas prise en compte dans les 16K de la mémoire écran. Elle sera donc ignorée durant le reste de cette étude.)

Si nous passions en mode 0, nous pourrions afficher 15 séries de caractères, chacune d'une couleur différente, en ajoutant PEN 4, PEN 5, etc. Nous pourrions imaginer PEN comme un moyen nous permettant de sélectionner des couleurs sur une palette limitée à un maximum de 16 couleurs (mode 0), de 4 couleurs (mode 1) ou de 2 couleurs (mode 2). Il est vrai que nous avons 27 couleurs dans notre boîte de peintures, mais nous sommes limités dans notre manière de les employer, car il faudrait 5 bits par pixel pour pouvoir sélectionner 27 couleurs (avec quelques combinaisons de réserve).

## L'ARTISTE AU TRAVAIL

La commande INK nous permet de changer les couleurs de notre palette. Elle indique au système laquelle des 4 couleurs nous voulons changer (premier paramètre) et quelle couleur nous allons mettre à la place (second paramètre). Quelle que soit la couleur que nous utilisions auparavant, elle est rangée dans la boîte. Ainsi, bien que nous soyons contraints à n'utiliser que 4 couleurs au maximum (en mode 1), nous pouvons choisir n'importe lesquelles parmi les 27 couleurs disponibles.

Essayez, par exemple :

```
INK 2,4
```

et la seconde série de chiffres (en fait, tout ce qui est écrit sur l'écran avec PEN 2 — le "stylo" 2) passera instantanément du turquoise vif au magenta.

N'importe laquelle des 4 "couleurs" peut être en réalité un jeu de 2 couleurs intermittentes. Essayez :

```
INK 1,20,8
```

et la première série de chiffres, qui a été écrite avec PEN 1, clignote en passant alternativement de la couleur 20 (turquoise vif) à la couleur 8 (magenta vif).

Il faut admettre que l'analogie de la palette commence maintenant à paraître un peu sommaire, puisqu'en changeant simplement l'encre de la palette nous modifions également la couleur de l'écran, pour peu qu'une petite partie de cette encre soit utilisée. L'artiste peint à la vitesse de l'éclair !

Un dernier mot d'explication sur la commande INK s'avère peut-être nécessaire. En mode 1, nous l'avons déjà vu, nous sommes limités à 4 couleurs. Elles sont définies avec PEN et dans le premier paramètre de la commande INK, où l'on peut leur donner une valeur allant de 0 à 3. Lorsqu'on fait démarrer le système, le papier sur lequel on "écrit" est défini par INK 0, et le "stylo" par INK 1. Dans le programme précédent, nous avons écrit une suite de chiffres en utilisant l'encre initiale des stylos — c'est-à-dire 1 — puis nous avons pris le stylo 2, puis le stylo 3. Nous n'avons pas modifié la couleur du papier, qui a conservé la couleur de l'encre 0. (Essayez donc de la changer. Par exemple, entrez simplement INK 0,6 et vous verrez dis-

paraître la troisième série de chiffres, puisque vous l'aurez alors écrite au stylo rouge sur du papier rouge.)

Tout comme nous pouvons poser un stylo pour en prendre un autre (la commande PEN), nous pouvons choisir un autre papier en utilisant la commande PAPER. PAPER 2 donne au "fond" de l'écran la couleur qui a été assignée à INK 2 pour toutes les applications ultérieures.

Il est important de comprendre qu'en entrant PAPER 2 ou PAPER 3, par exemple, vous ne modifiez pas ce qui se trouve déjà sur l'écran, mais seulement ce qui sera écrit par la suite, et donc que des instructions ultérieures comme INK 2,4 ou INK 0,6 auront un effet immédiat (et souvent spectaculaire).

## COULEURS LOGIQUES CONTRE COULEURS PHYSIQUES

Voici, à titre de référence, un tableau des numéros de couleur. Il devrait également vous aider à vous remémorer la différence entre les COULEURS LOGIQUES (le premier paramètre de INK) et les COULEURS PHYSIQUES (le second paramètre).

| COULEURS PHYSIQUES | | COULEURS LOGIQUES (PAR DÉFAUT) | | | | |
|---|---|---|---|---|---|---|
| Numéro de la couleur | Description de la couleur | Mode 0 | Mode 1 | Mode 2 | Papier | Stylo |
| 0 | Noir | 5 | | | | |
| 1 | Bleu | 0 | 0 | 0 | 0 | |
| 2 | Bleu vif | 6 | | | | |
| 3 | Rouge | | | | | |
| 4 | Magenta | | | | | |
| 5 | Mauve | | | | | |
| 6 | Rouge vif | 3 | 3 | | | |
| 7 | Pourpre | | | | | |
| 8 | Magenta vif | 7 | | | | |
| 9 | Vert | | | | | |
| 10 | Turquoise | 8 | | | | |
| 11 | Bleu ciel | | | | | |
| 12 | Jaune | 9 | | | | |
| 13 | Blanc | | | | | |
| 14 | Bleu pastel | 10 | | | | |
| 15 | Orange | | | | | |
| 16 | Rose | 11 | | | | |
| 17 | Magenta pastel | | | | | |
| 18 | Vert vif | 12 | | | | |
| 19 | Vert marin | | | | | |
| 20 | Turquoise vif | 2 | 2 | | | |
| 21 | Vert citron | | | | | |
| 22 | Vert pastel | 13 | | | | |
| 23 | Turquoise pastel | | | | | |
| 24 | Jaune vif | 1 | 1 | 1 | | 1 |
| 25 | Jaune pastel | | | | | |
| 26 | Blanc brillant | 4 | | | | |
| Intermittentes 1,24 | | 14 | | | | |
| Intermittentes 16,11 | | 15 | | | | |

Le tableau précédent montre que le papier ''par défaut'' est défini dans tous les modes par l'encre 0 et la couleur 1 (le bleu). De même, dans tous les modes, le stylo par défaut est défini par l'encre 1 et la couleur 24.

---

NOTE : En utilisant une couleur logique dont le numéro serait supérieur à ceux qui sont autorisés (par exemple avec PEN 5 en mode 1), vous n'introduirez pas d'erreur ; la valeur sera prise en considérant un reste sur 16 (en mode 0), sur 4 (en mode 1) ou sur 2 (en mode 2).

## EFFETS SPÉCIAUX

Ne vous imaginez pas, à partir des explications précédentes, que toutes les couleurs doivent toujours être différentes. Il peut parfois être utile de les rendre identiques. Par exemple, entrez :

```
INK 0,1
PAPER 0
INK 1,24
PEN 1
```

afin de donner un meilleur aspect à votre écran clignotant, puis relancez le programme, et entrez :

```
INK 1,1
```

La première série de chiffres disparaît aussitôt ! Et si nous entrons :

```
INK 1,23
```

elle réapparaît aussi rapidement, mais dans une couleur différente. Cela fait appel à une caractéristique très importante qui nous sera très utile par la suite, lorsque nous étudierons le graphisme — la possibilité de contrôler l'apparition et la disparition des images. Une utilisation correcte des commandes INK et PEN, associée à quelques

programmes de contrôle, nous permettra quelques "astuces" telles que :

- faire apparaître et disparaître "instantanément" des objets ;
- donner l'illusion qu'il y a plusieurs plans (ou plusieurs épaisseurs) en permettant aux objets du premier plan de passer "devant" des objets du second plan et de cacher le "fond" de l'écran ;
- animer des objets ou des personnages sur l'écran.

Nous en reparlerons plus tard ; il est temps de quitter la couleur pour passer au graphisme de l'Amstrad !

# 5

# TEXTE ET GRAPHISME

Poursuivant notre étude du chapitre précédent sur les couleurs de l'Amstrad, nous allons examiner maintenant les commandes concernant le texte et le graphisme, et les techniques permettant de gérer l'affichage.

## GRAPHIQUES DE BLOC

Tout le monde comprend le texte ; les caractères d'un texte sont ceux du tableau du code ASCII, allant du caractère 32 (l'espace) au caractère 127 (une sorte de damier réduit). Mais le graphisme a peut-être besoin d'une définition préliminaire. Avec l'Amstrad, nous avons deux sortes de graphiques, graphiques de bloc (ou mosaïques) et graphiques de ligne. Les graphiques de bloc sont de simples caractères qui sont affichés comme du texte. Par exemple :

```
PRINT CHR$(224)
```

affiche un graphique de bloc : un visage souriant. Puisque les graphiques de bloc se comportent exactement comme des caractères alphanumériques ordinaires, on peut définir ses propres graphiques si on le désire (voir Chapitre 3) et les combiner pour former des caractères plus larges.

L'Amstrad propose tout un ensemble de graphiques de bloc fort utiles, y compris des "jeux" de caractères appréciables quand on veut dessiner des cadres ou des diagrammes. Examinez les matrices agran-

dies du manuel (pages A3.2 à A3.13) et vous distinguerez plusieurs groupes :

| | |
|---|---|
| *Caractères 128-143* | Blocs. Chaque bloc représente 4 cellules, qui peuvent être ''allumées'' ou ''éteintes''. |
| *Caractères 144-159* | Lignes. Conçus pour relier le centre d'un caractère au milieu d'un bord, selon différentes combinaisons. |
| *Caractères 160-191* | Caractères complémentaires. Vous remarquerez divers caractères étrangers, comprenant des lettres grecques, des accents français, un point d'interrogation renversé de l'alphabet espagnol, etc. |
| *Caractères 192-255* | Symboles graphiques divers. |
| *Caractères 0-31* | Autres symboles divers. Comme il s'agit également de caractères de commande, vous devez les faire précéder de CHR$(1) si vous désirez les faire afficher sur l'écran (voir Chapitre 3), mais certains de ces symboles sont bien utiles, et il ne faut pas les oublier. |

Quand vous utilisez des graphiques de ligne normaux, n'oubliez pas qu'il peut être avantageux de les mêler à des graphiques de bloc judicieusement choisis, afin d'obtenir le meilleur effet.

## GRAPHIQUES DE LIGNE

Les graphiques de ligne sont, comme vous avez pu l'imaginer, formés de lignes. Initialisez le système et testez le petit programme suivant afin de dessiner un rectangle :

```
 5 CLS
10 MOVE 300,200
20 DRAW 340,200
30 DRAW 340,300
40 DRAW 300,300
50 DRAW 300,200
60 END
```

Les graphiques de ligne sont toujours dessinés sur un *écran graphique,* alors que les graphiques de bloc et les textes sont "dessinés" sur un *écran de texte.* Chaque type d'écran a ses propres limites et son propre curseur — mais le curseur graphique est toujours invisible ! Il existe malgré tout, et vous pouvez toujours trouver sa position selon des coordonnées de type XY, en utilisant les fonctions XPOS et YPOS. Après l'exécution du programme précédent, XPOS — la position horizontale — aura la valeur 300 et YPOS — la position verticale — aura la valeur 200. Essayez :

```
PRINT XPOS, YPOS
```

pour vérifier la position du curseur.

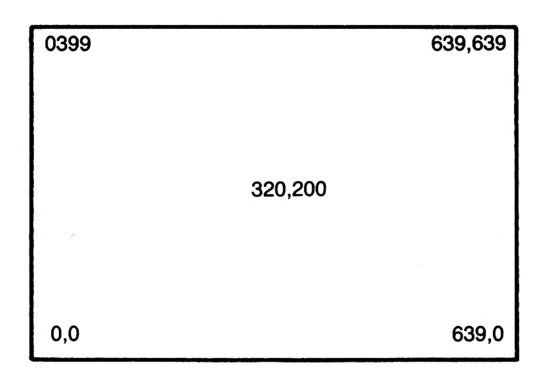
Les positions possibles de XPOS vont de 0 à 639, celles de YPOS vont de 0 à 399, car l'écran graphique a une définition de 640 × 400 points. Il n'y a en fait que 200 points possibles sur une verticale, mais en doublant ce chiffre la distance entre, disons, 10 points sera approximativement la même dans les deux directions, verticale ou horizontale, et cela facilitera les calculs lors de la conception des graphiques.

L'écran graphique conserve la même taille, quel que soit votre mode de travail ; par conséquent, les programmes graphiques écrits pour un mode donné fonctionneront aussi bien avec un autre mode, bien que les résultats puissent paraître très différents. En mode 0, le mode ayant la plus faible résolution, chaque pixel occupera 4 points horizontaux, et si vous entrez :

```
MODE 0
PLOT 320,200
PLOT 321,200
PLOT 322,200
PLOT 323,200
```

vous verrez qu'un seul point a été dessiné (ou plutôt, que 4 points ont été dessinés au même endroit !).

Les positions sur l'écran graphique sont toujours données en commençant par la coordonnée X. La position 0,0 représente un point situé en bas à gauche, et 639,399 un point situé en haut à droite.




Cette disposition n'est pas mauvaise, mais elle est inversée par rapport à celle utilisée par les écrans de texte. Ils comptent les lignes à partir de 1 (et pas 0) et partent de la position en haut à gauche jusqu'à la position en bas à droite, et la coordonnée X varie de 20 (en mode 0) à 80 (mode 2) en passant par 40 (en mode 1)... Enfin, cela permet de faire travailler nos méninges !

## COMMANDES GRAPHIQUES

Pour en revenir aux graphiques de ligne : les commandes sont simples et peu nombreuses. DRAW permet de dessiner une ligne — dans la couleur du "stylo graphique" — depuis la position du curseur graphique jusqu'à une nouvelle position. La couleur utilisée, en termes de graphisme, est celle que vous avez employée la fois précédente ! Si c'est la première fois qu'une commande graphique est utilisée, la couleur sera celle de l'encre n° 1.

Vous pouvez à tout moment sélectionner une nouvelle couleur graphique en ajoutant une couleur à la commande DRAW elle-même ; ainsi, DRAW 500,200,2 dessinera une ligne de la couleur de l'encre 2 (INK 2), quelle qu'elle soit, et toutes les commandes graphiques suivantes continueront d'utiliser l'encre 2 tant que vous n'aurez pas spécifié une autre encre.

MOVE positionne le curseur graphique à n'importe quelle position XY de l'écran, mais sans rien dessiner, et PLOT dessine un simple pixel (dont la taille variera selon le mode d'écran) à l'emplacement graphique spécifié. Comme DRAW, la commande PLOT peut recevoir pour troisième paramètre une couleur qui deviendra alors la nouvelle couleur graphique.

TEST peut être employé afin de savoir dans quelle couleur est dessiné un pixel particulier. Cela peut être très utile dans un programme de jeu, par exemple, lorsqu'on se déplace sur un écran qui a été parsemé d'objets divers de manière aléatoire.

DRAW, MOVE, PLOT et TEST utilisent tous des coordonnées XY absolues, mais il y a différents types de commandes qui utilisent des adresses relatives — c'est-à-dire relatives à la position actuelle du curseur. Ainsi, PLOTR 40,50 dessinera un point à l'emplacement XPOS + 40, YPOS + 50.

Dans le petit programme précédent, nous avons utilisé CLS pour effacer l'écran, mais à proprement parler cette commande efface les écrans de texte et nous aurions dû employer CLG, qui efface l'écran graphique. Dans le programme précédent, l'effet était le même, en supposant que vous ayez d'abord initialisé le système, mais ce ne sera pas toujours le cas. Normalement, CLG restaure la couleur définie par PAPER, quelle qu'elle soit (0 par défaut), mais vous pouvez redonner au fond de l'écran une couleur spécifique. CLG 2 donnera à l'écran la couleur de l'encre n° 2 qui pourra être ou non sa couleur par défaut (couleur 20 — turquoise vif), et qui restera la couleur de fond du graphisme tant que vous ne la changerez pas.

La seule commande du graphique de ligne qui reste à étudier est ORIGIN, qui nous permet de modifier l'emplacement de la position 0,0 (initialement en bas à gauche de l'écran).

Par exemple :

```
ORIGIN 320,200
```

placera le point 0,0 au centre de l'écran et il y restera tant que nous ne le déplacerons pas. Tant que ce ''point d'origine'' sera là, nous pourrons utiliser des coordonnées négatives pour nous déplacer à gauche ou en dessous du nouveau point 0,0. Par exemple, pour désigner le coin inférieur gauche de l'écran, nous pourrions entrer :

```
DRAW -320,-200
```

et

```
DRAW 319,-200
```

pour tracer une ligne jusqu'au coin inférieur droit.

Normalement, l'écran graphique occupe toute la superficie de l'écran physique (sauf la bordure qui ne fait partie ni de l'écran du texte, ni de l'écran graphique). Mais nous pouvons limiter le graphisme à une zone plus petite si nous le désirons, en utilisant une fois encore la commande ORIGIN.

```
ORIGIN 0,0,160,480,300,100
```

définira une fenêtre graphique réduite au centre de l'écran physique. Essayez d'entrer

```
CLG 2
```

pour observer les limites de cette nouvelle zone. Puis tapez CLS, en remarquant que l'effacement de l'écran de texte efface également l'écran graphique, car ils se superposent. CLG restaure l'écran graphique, et nous pouvons alors constater que le conflit entre les écrans obéit à une règle fort simple : c'est la dernière opération qui compte !

## DESSINER DES CERCLES

A part TAG et TAGOFF (voir plus loin), on n'emploie ici que les commandes graphiques. Nous ne disposons pas de CIRCLE ou de FILL, ce qui est dommage. Mais avec ces quelques commandes que nous venons d'étudier, il est possible de faire bien des choses, comme nous allons le voir.

Voici un sous-programme permettant de dessiner le bord d'un cercle.

```
1000 REM DESSIN D'UN CERCLE DE RAYON R EN CX,CY
1010 DEG
1020 ORIGIN CX,CY
1030 FOR J=1 TO 360
1040 PLOT R*COS(J),R*SIN(J)
1050 NEXT J
1060 RETURN
```

Modifions ainsi la ligne 1040 :

```
1040 MOVE 0,0 : DRAW R*COS(J), R*SIN(J)
```

si l'on veut obtenir un cercle plein.

Ce sous-programme considère implicitement (?) que l'origine est déplacée au centre du cercle. Si ce n'est pas ce que l'on souhaite, nous pouvons écrire :

```
1000 REM DESSIN D'UN CERCLE DE RAYON R EN CX,CY
1010 DEG
1020 FOR J=1 TO 360
1030 MOVE CX,CY
1040 PLOT R*COS(J),R*SIN(J)
1050 NEXT J
1060 RETURN
```

Ces deux sous-programmes doivent être appelés après avoir défini CX, CY et R. Par exemple :

```
10 CX= 320 : CY=200 : R=150 : GOSUB 1000
```

## UN SYSTÈME TOLÉRANT

Quand vous faites des essais avec différents cercles et d'autres formes, remarquez que si, accidentellement ou à dessein, vous voulez dessiner une ligne ou un point à l'extérieur de l'écran, le système calcule patiemment à quel endroit la ligne aurait abouti si l'écran avait été plus large, et qu'il fait de son mieux pour vous satisfaire. Essayez :

```
10 ORIGIN 0,0
20 MOVE 320,200
30 DRAW 0,200
40 DRAW 320,900
50 DRAW 1000,100
60 DRAW 320,200
```

afin de vérifier le résultat. Il vaut mieux cela qu'un arrêt du système accompagné d'un message d'erreur ! L'unique léger désagrément de cette tolérance est que, si vous vous trompez complètement dans vos calculs de coordonnées, des figures entières risquent d'être "dessinées" en dehors de l'écran, ce qui peut évidemment vous faire perdre beaucoup de temps.

En règle générale, si la commande graphique ne semble rien donner qu'un inquiétant silence, assurez-vous d'abord que vous ne des-

sinez pas dans la même couleur que le fond de l'écran, et vérifiez ensuite les valeurs actuelles de XPOS et YPOS pour être certain que vous ne vous êtes pas aventuré trop loin dans la jungle de l'outre-écran !

## DÉPLACER LA MÉMOIRE ÉCRAN

Dans le Chapitre 4, nous avons dit que la mémoire écran (la partie de la mémoire associée à l'écran) commence normalement à l'adresse &C000. Il s'agit d'une adresse par défaut, mais vous pouvez positionner la mémoire écran dans n'importe quelle plage de 16K si vous le désirez ; une routine du système d'exploitation, située en &BC08, permet de définir une nouvelle adresse pour la mémoire écran. En théorie, cette adresse pourrait aussi bien être 0 que &4000 ou &8000, mais en pratique il y a une excellente raison qui fait que seule l'adresse &4000 peut être utilisée comme adresse de remplacement. Si nous placions la mémoire écran en &0 ou en &8000, elle recouvrirait les zones allouées en permanence au système d'exploitation, et cela provoquerait inévitablement un blocage du système.

Si nous décidons de créer une nouvelle mémoire écran à partir de l'adresse &4000, nous pouvons également créer un dessin en &C000 et passer d'une adresse à l'autre pendant l'exécution du programme. C'est un moyen permettant de créer des effets spéciaux sur l'écran — par exemple de réaliser une animation. Exécutez le programme suivant :

```
5 REM PROGRAMME ROUES
10 GOSUB 1000
20 SCR=SCR XOR &80 : GOSUB 2000
30 GOTO 20
1000 REM INITIALISATION
1010 MEMORY 16383
1020 FOR J=0 TO 100
1030 READ X
1040 IF X=-1 THEN 1100
1050 POKE &8000+J,X
1060 NEXT J
1070 DATA &3E,&C0,&CD,&8,&BC,&C9
1080 DATA -1
1100 SCR=&C0 : GOSUB 2000
1110 START=1 : GOSUB 1900
1200 SCR=&40 : GOSUB 2000
1210 START=5 : GOSUB 1900
1299 RETURN
1900 CLG : DEG
1902 RESTORE 1995
1905 FOR WHEEL=1 TO 2
1907 READ GX,GY
```

```
1910 ORIGIN GX,GY : DEG
1920 FOR J=1 TO 360
1930 PLOT 50*COS(J),50*SIN(J),1
1940 NEXT J
1950 FOR J=START TO START+360 STEP 8
1960 MOVE 0,0
1970 DRAW 50*COS(J),50*SIN(J)
1980 NEXT J
1985 NEXT WHEEL
1990 RETURN
1995 DATA 160,100,480,100
2000 REM CHANGEMENT D'ECRAN
2010 POKE &8001,SCR
2020 CALL &8000
2999 RETURN
```

Les instructions DATA créent un sous-programme en assembleur qui charge dans le registre A la valeur &C0 — c'est-à-dire l'adresse supérieure de la mémoire écran — et qui appelle la routine du système d'exploitation située en &BC08. Le sous-programme 2000 s'assure que la valeur placée dans le registre A est soit &40, soit &C0.

Il existe un moyen de réaliser le même genre d'effet avec une seule zone de mémoire écran, mais en utilisant deux zones vous pouvez donner une dimension supplémentaire à des graphiques complexes. Apprécions la taille importante de la mémoire de l'Amstrad, qui nous permet d'utiliser 32K de mémoire écran et nous laisse encore de la place pour un grand programme en BASIC !

## LES ÉCRANS DE TEXTE – LES FENÊTRES

Oublions le graphisme pendant un moment et retournons au texte. Dans ce chapitre, nous avons déjà parlé d'un "écran de texte" et d'un "écran graphique". En fait, il y a 8 écrans de texte, mais ils sont tous placés par défaut dans la même zone, c'est-à-dire l'écran physique. Nous avons écrit sur l'écran 0, l'écran de texte par défaut, mais il en existe encore 7 autres, numérotés de 1 à 7. Il n'existe toutefois qu'un seul écran graphique.

La taille et la position d'un écran de texte peuvent être définies par la commande WINDOW, et c'est ainsi que nous pouvons faire coexister plusieurs écrans sur notre unique écran physique. A titre d'exemple, nous pourrions vouloir définir une fenêtre dans la partie supérieure droite de l'écran physique. Initialisons d'abord le système, puis tapons :

```
WINDOW #1,21,40,1,12
```

Cette instruction définit la zone de l'écran 1 (en mode 1) comme étant constituée des 20 colonnes les plus à droite et des 12 lignes supérieures.

Le fait qu'il y ait désormais deux fenêtres sur l'écran n'est pas immédiatement visible, mais essayez d'entrer et d'exécuter le programme suivant :

```
10 CLS
20 FOR J=1 TO 20
30 PRINT J;TAB(5);SQR(J)
40 NEXT
50 END
```

puis tapez la commande :

```
LIST #1
```

et le programme s'inscrira dans le coin supérieur droit sans affecter les chiffres placés à gauche, qui se trouvent bien évidemment sur l'écran 0.

Remarquez que CLS a été considéré par le système comme l'équivalent de CLS #0 et que l'écran 0 recouvre toujours l'ensemble de l'écran physique. En tapant :

```
WINDOW #0,1,20,1,25
```

vous réduirez l'écran 0 à la partie gauche de l'écran, et

```
WINDOW #2,21,40,13,25
```

établira une fenêtre dans la partie inférieure droite.

A partir de maintenant, chaque fenêtre sera nettement séparée des autres. Chacune conservera son emplacement de manière tout à fait indépendante, et l'affichage d'un texte dans une fenêtre n'affectera pas la position du curseur (invisible, sauf si le programme attend une réponse) dans les autres fenêtres. Une commande comme PAPER #1,3 ne modifiera que la couleur du fond d'un seul écran, et ensuite une commande CLS #3 provoquera un changement de couleur instantané. (Dans certaines applications, cela peut constituer un moyen rapide de colorer tout un rectangle d'un seul coup.)

N'oubliez pas que les fenêtres sont égoïstes et qu'elles n'hésitent pas à effacer leurs concurrentes s'il le faut — c'est la dernière appelée qui gagne !

Vous connaissez déjà les commandes CLS et PRINT (utilisez PRINT #3,... pour écrire sur l'écran 3). LOCATE déplace le curseur de texte vers la position XY spécifiée, mais souvenez-vous bien que dans les écrans de texte les positions se calculent différemment de celles de l'écran graphique — il faut partir du haut de l'écran et la première position est 1 (et pas 0).

Les fenêtres sont utilisées en particulier dans le programme de gestion personnelle décrit dans la cinquième partie.

## MÉLANGER GRAPHISME ET TEXTE

Normalement, lorsque vous affichez des caractères, le système prend l'emplacement du curseur comme position de début d'affichage et, lorsque l'affichage est terminé, il positionne le curseur derrière les caractères. Mais l'Amstrad vous permet d'écrire du texte sur l'écran graphique si vous le désirez. Après la commande TAG #1, tout affichage sur l'écran 1 débutera en partant de la position du curseur graphique et non plus de celle du curseur de texte. La position du curseur graphique sera avancée de 8 pixels chaque fois qu'un caractère sera affiché de cette manière.

La commande TAG est très utile pour afficher des titres de graphiques, de "camemberts" ou d'autres diagrammes. Pour revenir au mode d'affichage normal, employez la commande TAGOFF.

Il y a encore un problème lorsqu'on mélange du graphisme et du texte. Si vous voulez annoter un diagramme compliqué en utilisant la commande TAG suivie d'une commande PRINT, il pourrait être ennuyeux que l'affichage du texte efface une partie du graphisme. Cela se produira parce que, lorsqu'il affiche un caractère, le système efface normalement la matrice de 8 × 8 pixels et lui donne la couleur du fond (la couleur du papier) avant d'afficher le caractère dans la couleur du stylo. L'affichage du caractère est évidemment nécessaire, mais parfois nous aimerions bien que le système ne commence pas par effacer le fond.

Il est facile de résoudre ce problème ! Nous allons passer en "mode transparent". Cela signifie tout simplement que les caractères seront affichés sans que le fond soit d'abord effacé, exactement comme nous le souhaitons. La transparence est établie en entrant le code de com-

mande 22 suivi du paramètre 1 dans la fenêtre de notre choix — 0 par défaut, bien entendu.

```
PRINT CHR$(22)+CHR$(1);
```

validera la transparence sur l'écran 0 et

```
PRINT CHR$(22)+CHR$(0)
```

la désactivera de nouveau.

Vous verrez clairement quel effet cela produit si vous validez la transparence et que vous tapez ensuite :

```
LOCATE 1,1
LIST
```

à supposer bien sûr qu'un programme existe. Les lignes du programme paraissent embrouillées car tous les caractères qui se trouvaient déjà sur l'écran n'ont pas été effacés pour céder la place aux nouveaux caractères.

Nous reparlerons de la transparence dans le Chapitre 11.

## JEUX DE CARACTÈRES NATIONAUX

Tout comme elle peut être importante pour l'annotation des diagrammes, la transparence peut être très utile si nous voulons disposer des accents. Nous avons déjà remarqué que le jeu de caractères comprend les accents ; vous vous êtes peut-être demandé alors comment nous pourrions les utiliser sans effacer le caractère qui se trouverait déjà là. Maintenant, il devrait vous paraître évident que

```
PRINT "e";CHR$(8);CHR$(167)
```

affichera un e accent grave (si la transparence a été validée auparavant) et que

```
PRINT "u";CHR$(8);CHR$(160)
```

affichera un u accent circonflexe. Bien entendu, le CHR$(8) représente le retour arrière.

Vous pourriez donc établir vos propres caractères afin de disposer des caractères accentués, en utilisant peut-être le programme de DÉFINITION DES CARACTÈRES PAR L'UTILISATEUR du Chapitre 3, puis en faisant en sorte que les caractères accentués soit directement accessibles à partir du clavier, comme dans l'exemple suivant :

```
5 REM CARACTERES FRANCAIS.
10 SYMBOL AFTER 220
20 GOSUB 1000
30 GOSUB 2000
40 FOR J=228 TO 235:PRINT CHR$(J);:NEXT
999 END
1000 REM DEFINITION DES CARACTERES
1010 SYMBOL 228,&18,&18,&3C,&66,&7E,&60,&3C,0:REM E GRAVE
1020 SYMBOL 229,&C,&18,&C,&66,&7E,&60,&3C,0:REM E AIGU
1030 SYMBOL 230,&10,&18,&78,&C,&7C,&CC,&76,0:REM A GRAVE
1040 SYMBOL 231,&10,&38,&7C,&C,&7C,&CC,&76,0:REM A CIRCONFLEXE
1050 SYMBOL 232,&66,&0,&7C,&C,&7C,&CC,&76,0:REM A TREMA
1060 SYMBOL 233,&66,&0,&66,&66,&66,&66,&3E,0:REM U TREMA
1070 SYMBOL 234,&18,&3C,&0,&66,&66,&66,&3E,0:REM U CIRCONFLEXE
1080 SYMBOL 235,&0,0,&3C,&66,&60,&66,&3C,&18:REM C CEDILLE
1999 RETURN
2000 REM DEFINIR LES TOUCHES POUR L'ACCENTUATION
2010 KEY 135,CHR$(228)
2020 KEY 136,CHR$(229)
2030 KEY 137,CHR$(230)
2040 KEY 132,CHR$(231)
2050 KEY 133,CHR$(232)
2060 KEY 134,CHR$(233)
2070 KEY 129,CHR$(234)
2080 KEY 130,CHR$(235)
2999 RETURN
```

Essayez maintenant d'utiliser le pavé numérique. C'est formidable !

Si vous voulez imprimer vos résultats, il faudra néanmoins vérifier que votre imprimante dispose également des accents. C'est le seul ennui possible.

---

NOTE : Le *Guide de l'utilisateur* du CPC464 vous propose également une méthode pour disposer des accents (Appendice X, page A10.1).

# 6

# HI-FI PROGRAMMABLE !

Le synthétiseur de son intégré à l'Amstrad est un composant très élaboré qui nous offre :

- 3 canaux sonores et un canal de bruit ;
- 8 octaves ;
- le contrôle du volume et de la tonalité.

Pour profiter de toutes ces possibilités, il nous faut apprendre quelques commandes assez complexes. Elles sont peu nombreuses, mais il est indéniable qu'à première vue elles risquent de vous intimider. Ces commandes sont les suivantes :

```
SOUND
ENV
ENT
ON SQ() GOSUB
RELEASE
```

auxquelles s'ajoute une fonction :

```
SQ()
```

## LA COMMANDE SOUND

Dans ce chapitre, nous examinerons la commande SOUND : les autres commandes seront étudiées dans la deuxième partie. Grâce à cette commande, nous pouvons donner à l'Amstrad la "hauteur" de son qu'il devra produire, sa durée, le numéro du canal stéréo et le volume du son.

La commande SOUND peut recevoir d'autres paramètres, mais c'est déjà bien suffisant pour le moment !

Consultez l'Appendice VII du manuel d'utilisation de l'Amstrad, qui vous fournit la liste des fréquences avec les périodes correspondantes, et tapez l'instruction suivante pour avoir une idée du fonctionnement de la commande :

```
SOUND 1,284
```

Cela produit un *la* ayant une fréquence de 440 cps (cycles par seconde), également appelé "*la* international". Nous n'avons pas spécifié la hauteur proprement dite, mais la période (284) qui est calculée grâce à la formule présentée au bas de la page A7.3. Pour l'instant, il nous suffit de sélectionner nos notes à partir du tableau de l'Appendice VII.

Nous choisissons de faire sortir ce son par le canal 1, ou A. Nous n'avons spécifié ni la durée ni le volume, et les valeurs par défaut sont alors prises en compte — 1/5 de seconde pour la durée et 12 pour le volume. Répétez ce son en augmentant la durée :

```
SOUND 1,284,100
```

La durée est donnée en 1/100 de seconde, et cette fois le son a dû être produit pendant une seconde.

Testons maintenant le volume, dont la plage va de 0 (silence) à 7 (maximum) si l'on ne donne pas la valeur de *l'enveloppe du volume,* et de 0 à 15 si l'on utilise la commande ENV. Essayez :

```
10 FOR VOL=0 TO 7
20 SOUND 1,284,100,VOL
30 NEXT VOL
40 END
```

Nous pouvons également ajuster le son grâce à la molette qui se trouve sur le côté droit du boîtier de l'Amstrad.

Vous vous demandez peut-être comment le programme en BASIC peut s'achever bien avant que le son ne cesse. Nous expliquerons cela dans le prochain chapitre.

Au cas où vous seriez maintenant lassé d'entendre le *la* international, nous allons tenter de produire d'autres notes :

```
10 FOR PERIODE=3822 TO 16 STEP -30
20 SOUND 1,PERIODE,20,4
```

Affreux ! Une gamme très imparfaite ! L'ordinateur semble passer un temps fou dans les fréquences basses avant de glisser trop rapidement vers les fréquences élevées. Il nous faut manifestement trouver un meilleur moyen de monter ou descendre dans la gamme que celui qui consiste simplement à ajouter une valeur fixe à la période. Nous pourrions créer un tableau des périodes, mais utilisons plutôt une formule similaire à celle que donne le manuel :

```
  10 FOR OCTAVE= -3 TO 4
  20 FOR NOTE=0 TO 12
  30 GOSUB 1000
  40 SOUND 1,PERIODE,50,4
  50 NEXT NOTE
  60 SOUND 1,0,100,0
  70 NEXT OCTAVE
  80 END
1000 REM CALCULER LA PERIODE SELON OCTAVE ET NOTE
1010 FREQ=261.626*(2^(OCTAVE+NOTE/12))
1020 PERIODE=ROUND(125000/FREQ)
1030 RETURN
```

Voilà qui est mieux ! Cela paraît presque musical. Nous possédons maintenant une formule qui nous donnera la période en fonction de l'octave et de la note. C'est plus satisfaisant que de chercher sans cesse dans le tableau des périodes. Nous progressons !

## NOTRE PREMIÈRE MÉLODIE

Maintenant que nous avons les moyens de calculer n'importe quelle note de l'ensemble de la gamme, comment réaliser une mélodie ? Il nous faut donner au programme des informations sur les notes et leur durée, et sur la manière de produire les sons. Bien entendu, nous pourrions chercher les notes dans l'Appendice VII du manuel, les convertir en périodes et les entrer sous forme de données (avec DATA), mais puisque nous avons élaboré une formule, il semble préférable d'entrer les notes d'une manière qui se rapproche le plus possible de la notation musicale habituelle, en laissant à l'ordinateur le soin de calculer la période.

Pour les lecteurs qui auraient échappé à une formation musicale, les bases en sont expliquées dans la deuxième partie de cet ouvrage ;

peut-être devraient-ils poursuivre la lecture de ce chapitre, entrer la mélodie telle qu'elle est donnée, et y revenir plus tard lorsqu'ils auront saisi les éléments de la notation musicale.

Le programme suivant joue la mélodie du *God save the queen.* Les instructions DATA qui représentent les notes et leur durée sont placées dans les lignes 100 à 199. Le sous-programme 2000 convertit chaque note de la notation semi-musicale de l'instruction DATA pour lui donner un numéro d'octave et un numéro de note dans cette octave.

Le tempo est donné par un seul chiffre qui est utilisé pour calculer la durée de chaque note. Cela permet de modifier aisément le rythme de la mélodie en ne changeant qu'une valeur.

```
1 REM GOD SAVE THE QUEEN
5 TEMPO=20
7 ENT -1,1,1,3,2,-1,3,1,1,3
10 READ NOTE$,DUR$
15 IF NOTE$="" THEN END
20 GOSUB 2000
25 OCTAVE=OCTAVE+2
30 GOSUB 1000
40 SOUND 1,PERIOD,DUR,4,0,1
42 SOUND 1,PERIOD,1,0
45 GOTO 10
100 DATA C5,C,C5,C,D5,C,B4,DC,C5,Q,D5,C
105 DATA E5,C,E5,C,F5,C,E5,DC,D5,Q,C5,C
110 DATA D5,C,C5,C,B4,C,C5,C
115 DATA C5,Q,D5,Q,E5,Q,F5,Q,G5,C,G5,C,G5,C,G5,DC,F5,Q,E5,C
120 DATA F5,C,F5,C,F5,C,F5,DC,E5,Q,D5,C,E5,C,F5,Q,E5,Q,D5,Q,C5,Q
125 DATA E5,DC,F5,Q,G5,C,A5,Q,F5,Q,E5,C,D5,C,C5,DC
199 DATA "",""
1000 REM CALCULER PERIODE AVEC OCTAVE ET NOTE
1010 FREQ=261.626*(2^(OCTAVE+NOTE/12))
1020 PERIOD=ROUND(125000/FREQ)
1099 RETURN
2000 REM CALCULER NOTE, OCTAVE ET DUREE AVEC NOTE$ ET DUR$
2010 Z$=LEFT$(NOTE$,1)
2020 NOTE=INSTR("CCDDEFFGGAAB",Z$)-1
2030 IF NOTE<0 THEN OCTAVE=5:DUR=1:GOTO 2999
2040 Z$=RIGHT$(NOTE$,1)
2050 IF Z$="#" THEN NOTE=NOTE+1
2060 IF Z$="b" THEN NOTE=NOTE-1
2080 OCTAVE=VAL(MID$(NOTE$,2,1))-4:IF OCTAVE<0 THEN OCTAVE=1
2090 IF NOTE<0 THEN NOTE=0:OCTAVE=OCTAVE-1
2100 DUR=2^INSTR("HDSQCM",RIGHT$(DUR$,1))/16
2120 IF LEN(DUR$)=1 THEN 2200
2130 Z=INSTR("DT",LEFT$(DUR$,1))
2140 IF Z=0 THEN 2200
2150 IF Z=1 THEN DUR=1.5*DUR
2160 IF Z=2 THEN DUR=DUR/3
2200 DUR=DUR*TEMPO
2999 RETURN
```

Si vous pensez que le résultat ressemble à la prestation d'un violoniste ayant la tremblote, soyez patient. Les améliorations viendront plus tard !

Ce programme n'utilise pas la manière la plus efficace de produire des notes avec la commande SOUND, mais il fonctionne et sert de

démonstration au principe de création d'une mélodie. D'autres raffinements seront ajoutés dans la deuxième partie.

Le sous-programme 1000 convertit l'octave et la note en une période. Le tempo peut être ajusté en modifiant la ligne 5.

Différentes mélodies peuvent être jouées en transformant les lignes 100 à 125. La notation utilisée pour les notes est "XYZ" ; X représente la note (de *do* à *si*), Y représente le numéro de l'octave (de 1 à 8) et Z constitue un indicateur optionnel qui montre si la note doit être diésée ou non.

## AJUSTEMENT DU VOLUME

La tonalité est certes reconnaissable, mais elle est indéniablement mécanique. Néanmoins, on peut en dire autant d'un orgue de Barbarie ou d'une boîte à musique de l'époque victorienne. Le véritable défi est de rendre le son de l'ordinateur aussi musical que possible en tenant compte de ses inévitables limites. Ne vous découragez pas, nous venons à peine de commencer !

Notre *God save the queen* a été joué à un volume fixe. Vous avez vu que nous pouvons faire varier l'intensité du volume avec la commande SOUND. Cependant, nous ne pouvons pas le modifier à l'intérieur d'une seule commande, mais seulement d'une commande à l'autre. C'est la commande ENV qui nous permet de modifier le volume d'un son qui est joué, et cela fait beaucoup pour améliorer l'effet musical.

La commande ENV nécessite une explication approfondie — qui sera donnée ultérieurement — tout comme la commande ENT qui nous autorise à modifier la hauteur (c'est-à-dire la valeur de la période) pendant l'écoute d'un son. Un exemple en a été donné dans la mélodie précédente.

## RÉSUMÉ DES POSSIBILITÉS SONORES

Jusqu'à présent, nous n'avons fait qu'effleurer les possibilités sonores de l'Amstrad, mais nous pouvons espérer que vous avez déjà apprécié leur étendue. Dans la deuxième partie, nous poursuivrons notre étude pour examiner les variations de volume et de hauteur, les files d'attente (queues), les mélodies à deux ou trois voix séparées, la synchronisation et les effets spéciaux.

Toujours plus loin, toujours plus haut...

# II

# UN SYNTHÉTISEUR MUSICAL TRÈS COMPLET

Cette partie nous entraîne encore plus loin dans les mystérieuses possibilités sonores de l'Amstrad.

# 7

# LA NOTATION MUSICALE

Dans le dernier chapitre, nous avons établi les bases qui nous permettent de faire de la musique avec l'Amstrad, et nous avons présenté un programme pouvant jouer une mélodie simple. Puisqu'il y a 3 canaux musicaux, nous pouvons ajouter 2 voix supplémentaires afin de créer un son beaucoup plus satisfaisant.

Avant de faire cela , nous devons expliquer comment fonctionne l'écriture des sons sous une forme qui permette à d'autres personnes, n'ayant jamais entendu la mélodie, de reproduire les intentions du compositeur.

## LES ÉLÉMENTS DE LA NOTATION MUSICALE

Si vous regrettez de ne pas connaître la notation musicale, rassurez-vous. Si vous maîtrisez le BASIC, la notation hexadécimale ou le langage d'assemblage du Z80, la notation musicale vous paraîtra bien simple en comparaison !

La gamme musicale est, par convention, divisée en octaves dans lesquelles la fréquence de la note supérieure est exactement le double de la fréquence de la note inférieure. L'octave est divisée en 12 parties égales appelées demi-tons.

Généralement, dans la musique occidentale, toutes les notes sont sélectionnées à partir de ce 1/12 d'octave, et une gamme consiste à monter ou descendre l'échelle des demi-tons et des tons. Un ton représente deux demi-tons.

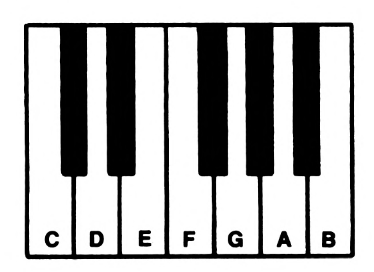
Bien évidemment, il n'existe pas de loi divine affirmant qu'une octave doit être divisée en 12 parties ; il s'agit simplement de notre convention, et dans d'autres parties du monde la musique utilise d'autres intervalles, comme les quarts de ton — que nous pourrions reproduire sur l'Amstrad si nous le voulions.

Une gamme composée de tous les demi-tons est appelée ''gamme chromatique'' :

```
1 REM GAMME CHROMATIQUE
10 READ OCTAVE, NOTE
20 IF OCTAVE=99 THEN END
30 GOSUB 1000
40 SOUND 1,PERIOD,20,4
50 GOTO 10
60 DATA 1,0,1,1,1,2,1,3,1,4,1,5,1,6,1,7,1,8,1,9,1,10,1,11,1,12,2,0
70 DATA 99,0
1000 REM CALCULER PERIODE AVEC OCTAVE ET NOTE
1010 FREQ=261.626*(2^(OCTAVE+NOTE/12))
1020 PERIOD=ROUND(125000/FREQ)
1099 RETURN
```

Si vous disposez d'un piano, vous devriez voir qu'une gamme chromatique se joue en commençant par une note et en frappant toutes les touches, noires et blanches, jusqu'à ce que vous soyez une octave au-dessus de la première note.

Chaque note de l'octave a reçu un nom. Les notes de l'octave représentées par les sept touches blanches sont appelées, selon les pays, *do, ré, mi, fa, sol, la, si,* ou encore C, D, E, F, G, A, B.



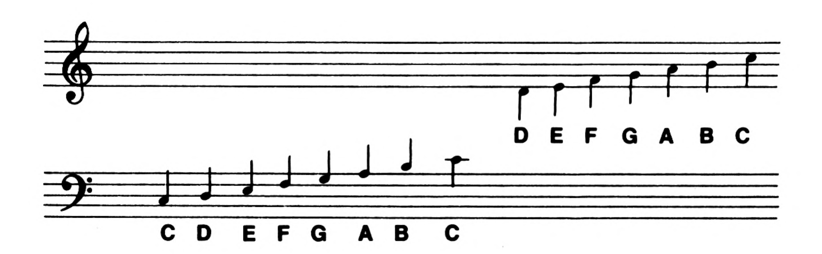

Vous remarquerez que la plupart des touches blanches sont situées à un ton entier des touches blanches voisines, c'est-à-dire qu'elles sont séparées par une touche noire dont le son se situe à un demi-ton de chacune des deux touches blanches voisines. Mais il n'y a qu'un demi-ton entre les touches *mi* et *fa*, et entre les touches *si* et *do*, et par conséquent une touche noire n'est pas nécessaire à cet emplacement.

Notre gamme chromatique débute avec la note *do* parce que nous avons utilisé la fréquence du *do* dans notre formule. Ainsi, *do* est la note 0. Pour jouer la gamme de *do* majeur, nous devons modifier ainsi l'instruction DATA :

```
60 DATA 1,0,1,2,1,4,1,5,1,7,1,9,1,11,1,12,2,0
```

Cette gamme correspond aux touches blanches du clavier du piano, et l'octave ainsi jouée commence avec le *do* "médian", qui se trouve à peu près au centre du clavier. Les touches noires donnent les demi-tons intermédiaires, et la notation # (dièse) ou ♭ (bémol) est placée juste après la note. # relève la note d'un demi-ton et ♭ la descend d'autant. Sur un clavier de piano, il n'y a pas de distinction entre un sol# et un la ♭ , par exemple.

Pour écrire notre mélodie, nous utilisons un papier sur lequel sont imprimées cinq lignes horizontales assez rapprochées, constituant ce que l'on appelle les portées musicales. Nous plaçons d'abord un signe au début de chaque portée pour indiquer quelle note sera représentée par telle ou telle ligne. Deux signes sont communément utilisés pour les instruments à clavier, mais il en existe d'autres pour des instruments comme l'alto. La clef de *sol* nous dit que la deuxième ligne, en partant du bas de la portée, représentera un *sol*, le *sol* qui se trouve au-dessus du *do* médian. Ainsi, le *do* médian se trouve juste en dessous de la portée, et si nous devons inscrire un *do* médian sur notre partition, il faudra placer la note sur un petit morceau de ligne supplémentaire appelée ligne ajoutée.




La clef de *fa* nous indique que la deuxième ligne en partant du haut de la portée représente la note *fa* (la *fa* qui précède le *do* médian) ; cette portée est donc mieux adaptée, pour nous présenter les notes basses, que la portée en clef de *sol*.

Si vous examinez une partition pour piano, vous verrez que la partie droite du clavier utilise généralement la clef de *sol*, qui est mieux adaptée pour l'écriture et la lecture des notes situées au-dessus du *do* médian ; inversement, la "main gauche" donne les notes plus basses et utilise la clef de *fa*. Cela n'est pas une règle absolue !

Nous disposons donc désormais d'une base permettant de déterminer la hauteur des notes, mais il faut également leur assigner une

durée. Si nous donnons à la triple croche la valeur 1, une double croche aura pour valeur 2, et ainsi de suite jusqu'à la ronde (32). Ici, la durée est une notion relative et non une valeur absolue ; la durée effectivement accordée à une note varie énormément selon le tempo de la pièce musicale, selon le jeu de l'interprète, etc.

| | | |
|---|---|---|
| 𝅘𝅥𝅰 | triple croche | 1 |
| 𝅘𝅥𝅯 | double croche | 2 |
| 𝅘𝅥𝅮 | croche | 4 |
| 𝅘𝅥 | noire | 8 |
| 𝅗𝅥 | blanche | 16 |
| 𝅝 | ronde | 32 |

La longueur d'une note peut être augmentée de moitié en la faisant suivre d'un point. Ainsi, une blanche pointée est l'équivalent d'une blanche et d'une noire, et a pour valeur 24.

## RÉSUMÉ

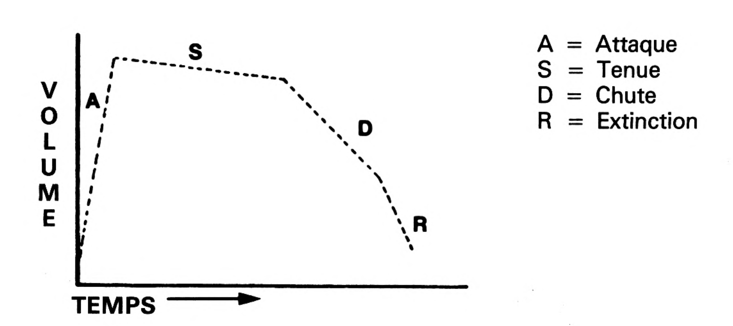
Maintenant que nous savons comment représenter la hauteur et la durée d'une note, nous pouvons revenir au sujet qui nous occupe : comment écrire des mélodies pour l'Amstrad. Bien·entendu, nous pourrions dire encore bien des choses à propos de la notation musicale ; nous n'avons pas parlé du rythme, du tempo, du phrasé, etc., mais un lecteur qui ne connaissait pas encore les partitions musicales devrait maintenant pouvoir suivre le développement qui suit ; cela pourrait l'encourager à étudier plus profondément ce qui peut devenir une véritable passion.

# 8

# VOLUME ET HAUTEUR D'UN SON

## LA COMMANDE ENV

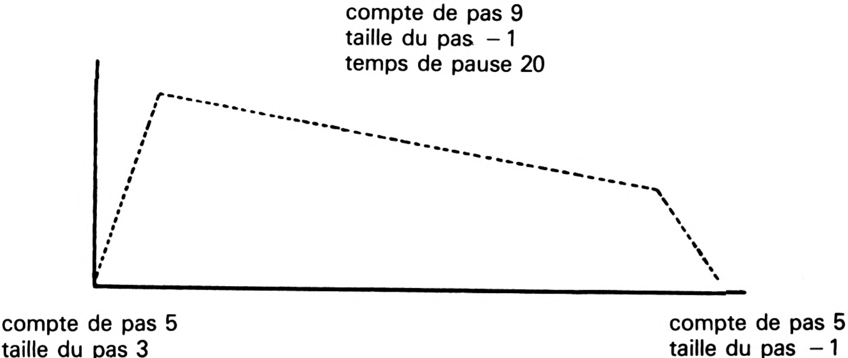
Si vous jouez une note sur un piano en maintenant la touche enfoncée pendant quelques secondes, la note commence par une augmentation rapide du volume, suivie par une période durant laquelle le volume baisse lentement. Finalement, le volume diminue et le son meurt rapidement lorsque vous relâchez la touche. Ce phénomène peut être représenté ainsi :




On appelle généralement les différentes phases : Attaque, Tenue, Chute et Extinction.

Si nous voulons produire électroniquement une note, nous pouvons donner des valeurs précises aux différentes phases, en spécifiant la durée de chaque phase et le volume "gagné" ou "perdu" ; autrement dit, si nous reprenons le diagramme précédent, nous devons donner la longueur des lignes et la valeur des angles.

La commande ENV nous permet une assez bonne simulation de cette action car nous pouvons définir jusqu'à 5 sections. Pour simuler une note de piano, nous pourrions définir 3 sections ; voici ce que pourrait être l'enveloppe d'une note ayant une durée de 2 secondes :




Elle pourrait être représentée par l'instruction :

**ENV 1,5,3,2,9,–1,20,5,–1,2**

dans laquelle le premier paramètre est le numéro de l'enveloppe (de 1 à 15) ; la première section (5,3,2) signifie : 5 pas ayant chacun une taille de valeur 3 et une durée de valeur 2 (en 1/100 de seconde) ; la deuxième section (9,–1,20) signifie : 9 pas de taille –1 et de durée 20 ; bien entendu, la troisième section (5,–1,2) signifie : 5 pas de taille –1 et de durée 2. Ici, la durée représente le "temps de pause" de chaque pas.

Évidemment, les pas sont en fait une suite de petits sauts "en escalier" plutôt qu'une ligne régulière, mais l'effet est à peu près identique dans la plupart des cas.

Pour utiliser la commande ENV, nous ajoutons un cinquième paramètre à la commande SOUND ; une commande SOUND correspondant à la commande ENV décrite précédemment pourrait être celle-ci :

**SOUND 1,284,200,0,1**

qui sélectionne 1 comme valeur de l'enveloppe de volume et 0 comme volume de départ. Bien que vous puissiez faire débuter le volume à n'importe quelle valeur (autorisée), il est souvent plus logique de le faire débuter à 0 si vous utilisez une commande ENV, ce qui permettra à toutes les manipulations du volume de se faire sur un seul paramètre.

Pour travailler sérieusement, il vaut mieux se donner la peine de construire le tableau suivant afin de vérifier les calculs :

| | NOMBRE DE PAS | TAILLE DU PAS | LONGUEUR DU PAS | LONGUEUR TOTALE | VOLUME FINAL |
|---|---|---|---|---|---|
| Volume de départ (commande SOUND) | | | | | 0 |
| Section 1 : | 5 | 3 | 2 | 10 | 15 |
| Section 2 : | 9 | −1 | 20 | 180 | 6 |
| Section 3 : | 5 | −1 | 2 | 10 | 1 |

Bien que la valeur du volume de départ donnée dans le tableau soit de 0, il faut savoir que la valeur du premier pas de la commande ENV est ajoutée au volume avant que le son ne soit produit ; la valeur du premier son entendu sera donc de 3 dans l'exemple précédent.

Vous pouvez aisément comprendre pourquoi il vous faut un crayon et du papier afin de déterminer correctement l'enveloppe de volume. Vous devez vérifier que la durée est exacte et que la valeur du volume n'est pas supérieure à 15 ou inférieure à 0. (Si vous dépassez ces limites, le système ne se bloquera pas ; le volume reviendra au départ et prendra une valeur entre 0 et 15, mais le résultat ne satisfera sans doute pas vos espérances.)

A propos, la durée indiquée dans la commande ENV a pour valeur 200, comme dans la commande SOUND, mais que se passerait-il s'il y avait une différence entre la durée spécifiée dans la commande SOUND et celle spécifiée dans la commande ENV ? En fait, c'est la valeur de la commande SOUND qui prime, mais il y a une ou deux exceptions. Examinons le tableau suivant :

*(Durée spécifiée dans)*

| SOUND | ENV | |
|---|---|---|
| 100 | 100 | Durée = 100, c'est-à-dire 1 seconde. |
| 90 | 100 | Durée = 90. La commande ENV n'est pas complètement exécutée. |
| 100 | 90 | Durée = 100. Le volume demeure constant durant les 10 derniers 1/100 de seconde. |
| 0 | 100 | Durée = 100. |
| −5 | 10 | La commande ENV est exécutée 5 fois ; la durée est donc de 50. |

Quand on écoute une seule note, l'effet est gâté par le ''cliquetis'' des changements de volume, mais en utilisant les changements de tons et plusieurs voix, nous pouvons insuffler un peu de vie dans une mélodie.

## MODIFICATION DE LA HAUTEUR — LA COMMANDE ENT

Tout comme la commande ENV est employée pour faire varier le volume, la commande ENT peut être utilisée pour changer de ton pendant que la note est jouée. Sa syntaxe est semblable à celle de la commande ENV et peut également contenir 5 sections. Chaque section comprend les paramètres suivants :

- nombre de pas,
- taille du pas,
- longueur du pas.

Une variation rapide de la fréquence constitue un vibrato. C'est une technique employée couramment par les instrumentistes et les chanteurs. Comme les marteaux du piano percutent les cordes, le pianiste ne peut pas réaliser de vibrato, contrairement aux joueurs d'instruments à cordes ou à vent.

L'Amstrad en est capable, bien qu'il faille admettre que le résultat est assez mécanique.

Pour le vérifier, chargez à nouveau *God save the queen* et entrez les modifications suivantes :

```
7 ENT -1,1,1,8,1,-1,8
40 SOUND 1,PERIODE, DUREE, 4,0,1
```

qui demandent au système d'utiliser l'enveloppe de ton n° 1 (le sixième paramètre de SOUND). Le premier paramètre de ENT donne à l'enveloppe de ton la valeur 1, mais cette valeur est négative pour indiquer que cette enveloppe doit être répétée pendant toute la durée de la note. Il y a ensuite deux sections : la première spécifie 1 pas de 1 période et d'une durée de 8/100 de seconde ; la seconde spécifie 1 pas de taille −1 et d'une durée identique à la première.

Cela donne un vibrato agréable et assez réaliste. Si vous n'êtes pas d'accord, faites d'autres essais. Modifiez un peu plus la hauteur :

```
7 ENT -1,1,2,8,1,-2,8
```

Cela paraît presque extrême pour les longues notes, mais c'est acceptable pour des notes courtes. (La fréquence acceptable d'un vibrato est liée à un jugement personnel !)

Remarquez que ces deux instructions ENT ne font varier le ton que ''vers le haut''. Pour obtenir un véritable vibrato, entrez :

```
7 ENT -1,1,1,5,2,-1,5,1,1,5
```

qui fait également varier le ton en dessous du ton initial. C'est un vibrato contenu qui devrait plaire à la plupart des oreilles.

En augmentant la valeur du temps de pause, le vibrato paraîtra lent et paresseux, alors qu'en la réduisant vous obtiendrez un effet plus ''tendu'', jusqu'au moment où vous jugerez sans doute que le son devient réellement névrotique !

## C'EST POUR VOUS !

La commande ENT n'est pas seulement utile en musique. Entrez :

```
  10 ENT -1,1,1,2,1,-1,2
  20 GOSUB 1000
  30 GOSUB 1000
  40 SOUND 1,0,200,0
  50 GOTO 20
1000 REM
1010 SOUND 49,70,50,15,0,1
1020 SOUND 28,35,50,5,0,1
1030 SOUND 42,17,50,5,0,1
1040 SOUND 1,0,15,0
1999 RETURN
```

Un observateur attentif peut remarquer que le son produit par le programme précédent n'est pas absolument correct, car la période de la ligne 1030 devrait avoir 17.5 comme valeur idéale. Mais la plupart des téléphones sont un peu ''désaccordés'' ; nous n'aurons donc aucun repentir !

Voilà un bon moyen d'améliorer vos programmes en les ''embellissant'' grâce aux commandes sonores !

# 9

# HARMONIE A DEUX ET TROIS VOIX

Lorsque nous avons expliqué comment faire pour programmer une mélodie, dans le Chapitre 6, il a été spécifié que la méthode présentée n'était pas la plus efficace. Dans ce chapitre, nous examinerons une facette du BASIC de l'Amstrad qui permet de programmer beaucoup plus aisément des airs musicaux.

## LES QUEUES

En utilisant la commande SOUND, vous remarquerez qu'un programme en BASIC qui possède plusieurs instructions sonores se termine avant la mélodie — il paraît capable de ''distancer'' l'exécution, en quelque sorte.

Et c'est effectivement ce qui se produit. En fait, la commande SOUND place les instructions dans une queue (une file d'attente) sonore, et le système d'exploitation prend les sons qui se trouvent au début de la queue pour les donner, en temps voulu au synthétiseur sonore.

La queue semble contenir 4 sons et le programme en BASIC la remplit très vite, puis attend que le premier son soit joué et que les autres sont soient ''avancés'' dans la queue par le système d'exploitation, qui donne le deuxième son au synthétiseur. Le BASIC peut alors placer la cinquième note dans la queue, et attend impatiemment de pouvoir introduire le sixième son.

Cela paraît bien être une perte de temps, et le BASIC de l'Amstrad est conçu de manière à nous éviter une attente inutile. Si le système sait que nous voulons être informés du moment où une place sera libre dans la file d'attente, nous pouvons aller tranquillement prendre un bain de soleil en sachant que le BASIC viendra nous réveiller avec une tasse de thé fumante quand tout sera prêt. En fait, il s'agit d'une interruption semblable à celle que nous avons pu étudier dans le Chapitre 2, avec la commande AFTER

Donc, si nous envoyons des notes par le canal 1, nous précisons :

```
ON SQ(1) GOSUB 3000
```

et chaque fois qu'une place est libre dans la queue du canal 1, notre programme sera interrompu et passera la main au sous-programme 3000, qui glissera une autre note dans la queue grâce à une commande SOUND avant de rendre le contrôle (avec RETURN) au programme principal, qui pourra poursuivre sa tâche un moment interrompue.

Avant que le sous-programme n'atteigne RETURN, nous devons normalement lui proposer une autre instruction ON SQ(1) GOSUB 3000 pour valider à nouveau le mécanisme d'interruption. En supposant, bien sûr, qu'il y ait d'autres notes à jouer. C'est simple !

Cette possibilité est particulièrement utile lorsque nous ajoutons de la musique pour soutenir une autre tâche de l'ordinateur, par exemple un jeu. Apparemment, le BASIC de l'Amstrad est le seul à posséder une telle faculté.

L'exemple suivant illustrera ce que nous venons d'étudier :

```
  10 ON SQ(1) GOSUB 3000
  20 GOTO 20

3000 REM AJOUT AU CANAL 1
3010 SOUND 1,284+J*8,300,4
3020 J=J+1
3030 PRINT J
3040 ON SQ(1) GOSUB 3000
3050 RETURN
```

Vous verrez que les premiers nombres sont affichés très rapidement pendant que les premiers sons se placent "à la queue leu leu" dans la file d'attente ; ensuite, l'affichage est beaucoup plus lent. Bien entendu, dans un véritable programme, la ligne 20 accomplirait une tâche utile au lieu de faire une simple boucle sur elle-même.

## SYNCHRONISATION

Jusqu'à présent, nous n'avons utilisé qu'un seul canal, le canal 1 ou A. Augmentons nos exigences et réquisitionnons les canaux B et C :

```
10 SOUND 1,284,200
20 SOUND 2,338,200
30 SOUND 4,426,200
```

Remarquez que le canal C n'a pas la valeur 3, comme vous pourriez vous y attendre, mais 4, car le premier paramètre de SOUND représente un "statut du canal" associé à un bit significatif, et :

- 1 valide le bit 0 : dirige le son vers le canal A,
- 2 valide le bit 1 : dirige le son vers le canal B,
- 4 valide le bit 2 : dirige le son vers le canal C,
- 8 valide le bit 3 : synchronisation avec le canal A, etc.

L'accord ainsi déterminé (un accord parfait en *ré* majeur) semble débuter par une "frappe" simultanée des 3 notes, mais cette impression est simplement due au fait que le BASIC est rapide. Insérez :

```
25 FOR J=1 TO 500 : NEXT J
```

et vous comprendrez le problème. Mais nous pouvons éviter cet ennui en profitant de la faculté de synchronisation.

Normalement, lorsqu'un son est placé dans la queue par la commande SOUND, il est joué dès qu'il parvient au sommet de la queue. Mais vous pouvez lui associer des bits supplémentaires, ou des "indicateurs", qui modifieront tout cela.

Si vous envoyez un son vers le canal A en lui demandant de se synchroniser avec le canal B, le son attendra sagement dans la queue jusqu'à ce qu'un son soit arrivé au sommet de la queue du canal B — auquel on aura demandé de se synchroniser avec le canal A ! Alors, et alors seulement, les queues des deux canaux libéreront leur note respective et produiront ainsi un accord merveilleusement synchronisé.

Essayez :

```
SOUND 33,284,200
```

Il ne se passe rien. Nous avons entré un son dans la file d'attente, mais nous avons validé le bit 5 et le bit 1 du premier paramètre ; or,

la validation du bit 5 signifie ''synchronisation avec le canal C'' (dans le manuel de l'Amstrad, on parle également de ''rendez-vous'' ; voir page 6.5).

Si nous tapons maintenant :

**SOUND 4,426,200**

nous entendons une note jouée par le canal C, mais il n'y a encore rien sur le canal A car il n'était pas précisé que la note envoyée vers le canal C devait être synchronisée avec une note du canal A. Nous devons entrer :

**SOUND 12,426,200**

et cette fois nous entendons deux notes, car la valeur ''12'' valide les bits 2 et 3, et cela signifie ''envoyer la note vers le canal C en effectuant une synchronisation (un rendez-vous) avec le canal A''.

La même méthode peut être utilisée avec le canal B et nous pouvons diriger des sons vers différents canaux à différents moments en sachant qu'ils seront joués simultanément. Quel soulagement !

Ce n'est pas seulement vital pour la première note d'une pièce musicale, mais également pour les accords ultérieurs. Dans la plupart des pièces, les différentes voix produiront des notes de durées différentes, et le canal A peut vouloir jouer 32 notes pendant que le canal B en joue 8 et que le canal C en joue 6 ! Ainsi, certaines notes seront synchronisées et d'autres ne le seront pas.

## UN CADRE POUR L'HARMONIE

Forts de notre nouveau savoir, nous allons concevoir un cadre qui nous permettra de réaliser des airs à deux ou trois voix en utilisant la synchronisation et les queues sonores. La mélodie choisie pour vérifier notre cadre est une bourrée de Haendel

```
5 REM COPYRIGHT (C) 1984 JOHN BRAGA
10 REM JOUER UNE MELODIE DE 1 A 3 VOIX
15 TRUE=-1:FALSE=0:DONE(1)=TRUE:DONE(2)=TRUE:DONE(4)=TRUE
20 NUMVOICE=3
30 SYNCHA=8 : SYNCHB=16:SYNCHC=32
40 TEMPO=4
50 GOSUB 1000 : REM INITIALISATION
60 ON 4-NUMVOICE GOTO 70,80,90
GOSUB 70 ON SQ(4) GOTO 4100
GOSUB 80 ON SQ(2) GOTO 4050
```

GOSUB

```
90 ON SQ(1) GOTO 4000
100 IF DONE(1) AND DONE(2) AND DONE(4) THEN 110 ELSE 100
110 INPUT "ENCORE UNE FOIS ";Z$
120 IF UPPER$(Z$)<>"O" THEN 200
130 DONE(1)=0:DONE(2)=0:DONE(4)=0:NOTE(1)=0:NOTE(2)=0:NOTE(4)=0:GOT
O 60
200 END
1000 REM INITIALISATION
1010 DIM PERIOD(4,200),DUR(4,200),SYNCH(4,200)
1020 FOR J=1 TO NUMVOICE
1030 CHANNEL=2^(J-1)
1040 NUMNOTES=0
1050 DONE(CHANNEL)=FALSE
1060 IF CHANNEL=1 THEN RESTORE 5000
1070 IF CHANNEL=2 THEN RESTORE 5400
1080 IF CHANNEL=4 THEN RESTORE 5700
1100 READ NOTE$,DURATION,SYNCH$
1110 IF NOTE$="" THEN 1500
1120 GOSUB 2000 : REM CALCUL DE PERIODE AVEC NOTE$
1200 NUMNOTES=NUMNOTES+1
1210 PERIOD(CHANNEL,NUMNOTES)=PERIOD
1220 DUR(CHANNEL,NUMNOTES)=DURATION*TEMPO
1230 SYNCH(CHANNEL,NUMNOTES)=SYNCH
1240 GOTO 1100
1500 PERIOD(CHANNEL,0)=NUMNOTES
1510 NEXT J
1600 ENT -1,1,1,7,1,-1,7
1999 RETURN
2000 REM CALCUL D'OCTAVE ET NOTE AVEC NOTE$
2010 Z$=LEFT$(NOTE$,1)
2020 NOTE=INSTR("C D EF G A BR",Z$)-1
2025 IF NOTE=12 THEN OCTAVE=0:GOTO 2100
2030 IF NOTE<0 THEN OCTAVE=5 : DUR=1 : GOTO 2999
2040 Z$=RIGHT$(NOTE$,1)
2050 IF Z$="#" THEN NOTE=NOTE+1
2060 IF Z$="b" THEN NOTE=NOTE-1
2080 OCTAVE=VAL(MID$(NOTE$,2,1))-3 : REM SI OCTAVE<0 ALORS OCTAVE=1
2090 IF NOTE<0 THEN NOTE=0 : OCTAVE=OCTAVE-1
2100 GOSUB 3000
2110 SYNCH=0
2120 FOR J1=1 TO LEN(SYNCH$)
2130 IF MID$(SYNCH$,J1,1)="A" THEN SYNCH=SYNCH+SYNCHA
2140 IF MID$(SYNCH$,J1,1)="B" THEN SYNCH=SYNCH+SYNCHB
2150 IF MID$(SYNCH$,J1,1)="C" THEN SYNCH=SYNCH+SYNCHC
2160 NEXT J1
2999 RETURN
3000 REM CALCULER PERIODE AVEC OCTAVE ET NOTE
3005 IF NOTE=12 THEN PERIOD=0 : GOTO 3999
3010 FREQ=261.626*(2^(OCTAVE+NOTE/12))
3020 PERIOD=ROUND(125000!/FREQ)
3999 RETURN
4000 CHANNEL=1
4040 GOTO 4200
4050 CHANNEL=2
4090 GOTO 4200
4100 CHANNEL=4
4200 NOTE(CHANNEL)=NOTE(CHANNEL)+1
4205 VOL=5
4207 TONE=0
4210 SOUND CHANNEL+SYNCH(CHANNEL,NOTE(CHANNEL)),PERIOD(CHANNEL,NOTE
(CHANNEL)),DUR(CHANNEL,NOTE(CHANNEL)),VOL,0,TONE
4220 IF NOTE(CHANNEL)>=PERIOD(CHANNEL,0) THEN DONE(CHANNEL)=TRUE:GO
TO 4999
4300 J=CHANNEL
4310 IF J>3 THEN J=3
4320 ON J GOTO 4400,4420,4440
4400 ON SQ(1) GOSUB 4000
4410 GOTO 4999
4420 ON SQ(2) GOSUB 4050
4430 GOTO 4999
4440 ON SQ(4) GOSUB 4100
4999 RETURN
```

```
5000 REM DONNEES POUR LE CANAL A
5010 DATA D5,6,C,R,2,,D5,7,C,R,1,,B4,6,C,R,2,,C5,4,C,B4,4,,A4,4,C,G
4,3,,R,1," "
5015 DATA E5,6,C,R,2,,G5,14,C,R,2,,F5#,4,C,E5,3,,R,1," "
5020 DATA D5,5,C,R,3,,C5,4,C,B4,3,,R,1,,A4,4,C,B4,4,,C5,4,C,A4,3,,R
,1," "
5025 DATA B4,6,C,R,2,,G4,14,C,R,2,,A4,6,C,R,2," "
5030 DATA B4,4,C,C5#,4,,D5,4,C,B4,3,,R,1,,C5#,4,C,D5,4,,E5,4,C,C5#,
3,,R,1," "
5035 DATA D5,4,C,E5,4,,F5#,4,C,D5,3,,R,1,,E5,4,C,F5#,4,,G5,4,C,E5,3
,,R,1," "
5040 DATA F5#,4,C,G5,4,,A5,4,C,R,4,,A4,4,C,R,4,,C5#,4,C,R,4,,D5,20,
C,R,4," "
5050 DATA A5,6,C,R,2,,A5,8,C,F5#,6,C,R,2,,G5,4,C,F5#,4,,E5,4,C,D5,3
,,R,1," "
5055 DATA G5,6,C,R,2,,B5,14,C,R,2,,E5,7,C,R,1," "
5060 DATA D5#,8,C,E5,8,C,F5#,7,C,R,1,,G5,4,C,A5,3,,R,1," "
5065 DATA G5,6,C,R,2,,E5,14,C,R,2,,D5,14,C,R,2," "
5070 DATA C5,4,C,B4,3,,R,1,,C5,5,C,R,3,,C5,14,C
5075 DATA R,2,,B4,4,C,A4,3,,R,1,,B4,6,C,R,2,,D5,6,C,R,2," "
5080 DATA E5,4,C,F5#,4,,G5,4,C,E5,3,,R,1,,F5#,4,C,G5,4,,A5,4,C,F5#,
3,,R,1," "
5085 DATA G5,4,C,A5,4,,B5,4,C,G5,3,,R,1,,A5,4,C,B5,4,,C6,4,C,A5,3,,
R,1," "
5090 DATA B5,5,C,R,3,,A5,4,C,G5,3,,R,1,,F5#,4,C,G5,4,,A5,4,C,F5#,3,
,R,1," "
5095 DATA G5,4,C,A5,4,,B5,4,C,G5,3,,R,1,,A5,6,C,R,2,,C6,6,C,R,2," "
5100 DATA B5,6,C,R,2,,A5,4,C,G5,3,,R,1,,F5#,4,C,G5,4,,A5,4,C,F5#,3,
,R,1," "
5105 DATA G5,4,C,A5,4,,B5,4,C,G5,3,,R,1,,A5,4,C,B5,4,,C6,4,C,A5,3,,
R,1," "
5110 DATA B5,4,C,C6,4,,D6,7,C,R,1,,B5,7,C,R,1,,A5,4,C,G5,4," "
5115 DATA R,2,BC,R,6,,G5,24," "
5399 DATA "",0," "
5400 REM DONNEES POUR LE CANAL B
5410 DATA R,2,AC,R,3,,B3,27," "
5699 DATA "",0," "
5700 REM DONNEES POUR LE CANAL C
5705 DATA G3,5,A,F3#,7,A,G3,5,A,D4,6,A,E4,5,A
5710 DATA C3,6,A,C4,5,A,R,3,,B3,4,,C4,5,A
5715 DATA F3#,6,A,G3,6,A,C3,6,A,D3,6,A,G2,6,A,G3,14,A,D3,6,A
5720 DATA G3,6,A,F3#,6,A,E3,6,A,A3,5,A,F3#,6,A,D4,5,A,C4#,6,A,A3,5,
A,D4,4,A,F3#,4,A,A3,4,A,A2,4,A,D3,8,A,D4,4,,C4#,4,,D4,6," "
5725 DATA D3,5,A,D4,7,A,A3,5,A,D4,6,A,C4,5,A
5730 DATA B3,5,A,G3,4,A,A3,4,,B3,8,,C4,7,A
5735 DATA B3,8,A,E3,8,A,A3,8,A,B3,7,A
5740 DATA E3,4,A,R,4,,E4,14,A,B3,6,A,R,2," "
5745 DATA E3,8,,G3#,8,A,A3,6,A,A2,6,A,R,2," "
5750 DATA D3,8,,F3#,8,A,G3,6,A,B2,4,A
5755 DATA C3,4,A,A3,4,A,D4,4,A,C4,4,A
5760 DATA B3,4,A,G4,4,A,F4#,4,A,D4,4,A
5765 DATA G3,4,A,B3,4,A,D4,4,A,D3,4,A
5770 DATA E3,4,A,C4,4,A,F3#,4,A,D4,4,A
5775 DATA G3,4,A,B3,4,A,D4,4,A,C4,4,A
5780 DATA B3,4,A,G3,4,A,F3#,4,A,D3,4,A
5785 DATA G3,5,A,B3,5,A,D4,5,A,D3,5,A
5790 DATA R,2,AB,G3,30," "
5899 DATA "",0," "
```

## COMMENTAIRE SUR LA BOURRÉE

La partie principale du programme occupe les lignes 10 à 200. Certaines variables sont initialisées et un saut est accompli jusqu'au sous-programme 1000 où sont définis les tableaux qui contiennent les notes.

Ensuite, les sous-programmes d'interruption sont amorcés par les lignes 70-90 et le programme fait une boucle à la ligne 100 jusqu'à ce que toutes les voix aient achevé leur partition.

Le sous-programme 1000-1999 définit 3 tableaux, PERIODE, DUREE et SYNCH, qui contiennent les valeurs de chaque note pour chaque voix. Les valeurs représentent évidemment la période, la durée et la synchronisation. Les notes sont lues par les instructions DATA commençant à la ligne 5000 et le sous-programme 2000-2999 est utilisé pour convertir une note en une période. Les instructions RESTORE des lignes 1060-1080 sont superflues si les 3 voix sont utilisées, mais sont utiles au cas où voüs faites jouer un seul canal (voir plus loin).

Le sous-programme 3000-3999 est appelé par le sous-programme 2000.

Le sous-programme 4000-4999 contient des instructions qui sont appelées par le BASIC chaque fois qu'il y a une place vide dans une queue. Il contient la commande SOUND qui prend les notes des tableaux (définis en 1000-1999) pour les placer dans les queues. Remarquez que le sous-programme d'interruption musicale doit se réinitialiser avant de donner le contrôle au programme principal avec la commande RETURN.

Les instructions DATA débutent en 5000 et sont divisées en trois sections, une section pour chaque voix. Remarquez l'utilisation de R pour créer un "reste", c'est-à-dire une période de silence.

## INTERPRÉTATION

Il est fascinant de voir comment on peut modifier l'interprétation de la bourrée (et bien entendu de n'importe quelle autre mélodie) en effectuant simplement de petits changements parmi les instructions DATA. Nous allons présenter quelques-uns de ces changements ; bien d'autres possibilités vous sont offertes.

Vous vous demandez peut-être pourquoi les instructions DATA du programme contiennent tant de restes, et pourquoi les valeurs de durée des noires vont de 4 à 8. La réponse est que certaines notes doivent être jouées *staccato* (brèves et bien détachées) et qu'elles ont des durées plus courtes. La valeur entière est complétée par un reste dans le cas du canal A, mais ce n'est généralement pas nécessaire pour le canal C car la synchronisation oblige de toute manière ce canal à "attendre" le canal A.

Des restes de faible valeur sont utiles pour que la note suivante paraisse "articulée" plutôt que "liée". Avec le programme précédent, il est possible de réaliser de nombreuses expériences sur la durée des notes et sur les effets des restes. Vous pouvez nettement transformer l'interprétation si vous le désirez.

Une enveloppe de ton est utilisée par les canaux B et C, mais pas par A, car nous pensons que le plus petit vibrato réalisable est excessif sur des notes élevées. On aimerait, de manière idéale, pouvoir réaliser un vibrato avec des valeurs inférieures à une période dans le registre supérieur, et ce n'est malheureusement pas possible. Mais n'hésitez pas à travailler sur la commande ENT de la ligne 1600 et sur l'instruction de la ligne 4207. Essayez :

```
4207 TONE=0
```

pour ne pas avoir de vibrato, puis :

```
4207 IF CANAL=4 THEN TONE=1 ELSE TONE=0
```

pour un vibrato sur le canal bas uniquement, puis :

```
4207 TONE=1
```

pour un vibrato sur les troix canaux — ou modifiez complètement la commande ENT !

Testez aussi le contrôle de volume de l'instruction 4205. Il n'y a pas de commande ENV, mais vous pouvez essayer de donner une prédominance à la voix supérieure si vous le désirez. Essayez :

```
4205 IF CANAL=1 THEN VOL=5 ELSE VOL=4
```

ou quelque chose de ce genre.

Remarquez comment est définie la dernière mesure (lignes 5115, 5410 et 5790). Il y a au début de la mesure un petit reste synchronisé qui, à proprement parler, n'est pas dans le tempo. Cela donne l'effet d'un léger glissement. Ensuite, les notes sont légèrement "décalées" à partir des basses pouur produire un effet d'arpège. Ce style est particulièrement courant dans la musique pour clavecin, et cette bourrée a été écrite pour le clavecin. Enfin, la dernière note a une durée plus longue que prévu. Il ne s'agit pas d'une erreur !

Bien entendu, le programme précédent peut être utilisé comme base pour créer de nombreuses pièces musicales. Dans la plupart des cas, il suffira de modifier les instructions DATA des lignes 5000-5399 (canal A), 5400-5699 (canal B) et 5700-5899 (canal C).

Créer un air est un processus laborieux et il vaut mieux l'accomplir par étapes. Chargez d'abord le programme de bourrée et supprimez toutes les instructions DATA, à l'exception des instructions "finales" (lignes 5399, 5699 et 5899).

Entrez d'abord les instructions DATA du canal A, puis effectuez les modifications suivantes avant d'exécuter le programme :

```
20 NUMVOIX=1
30 REM ...
```

Cela supprime effectivement la synchronisation. Vérifiez "à l'oreille" la valeur et la durée des notes. Testez de petits restes pour obtenir l'articulation voulue. Vous verrez que votre tâche sera considérablement simplifiée si vous employez des instructions DATA plutôt courtes — généralement, une instruction par mesure ; cela facilitera la recherche des erreurs. Vous pouvez modifier 2 instructions qui servent à la vérification de la mélodie :

```
1105 PRINT NOTE$;" ";
1505 PRINT
```

Entrez ensuite la "partition" des basses. Il n'est pas très utile d'exécuter cette partie toute seule, car elle doit être synchronisée avec le canal A, et elle paraîtra plutôt bizarre si vous avez placé une commande REM à la ligne 30. Mais rendez aux lignes 20-30 leurs valeurs correctes et entrez :

```
4205 IF CANAL=4 THEN VOL=4 ELSE VOL=0
```
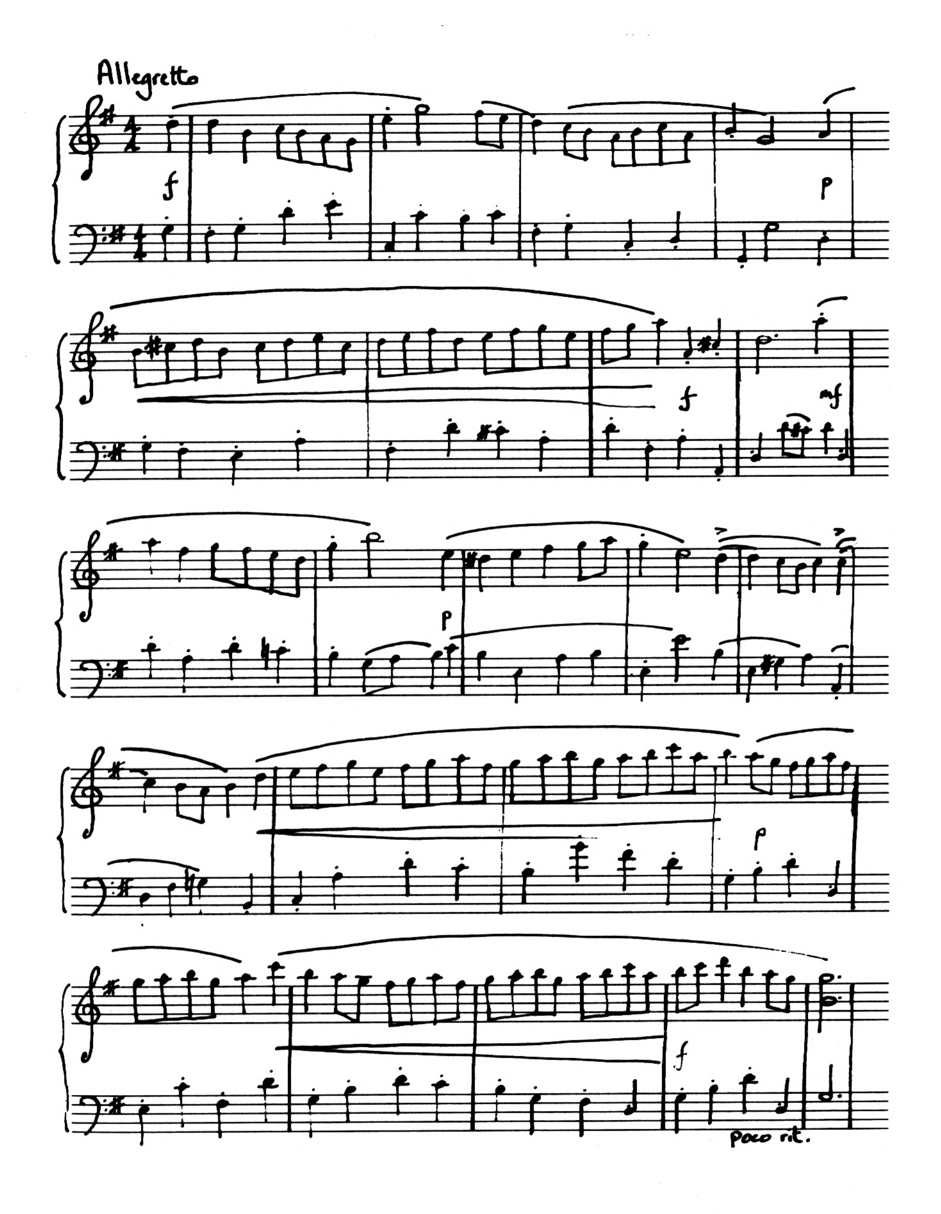
qui permettra de n'entendre que les basses. Une fois encore, vous pouvez jongler avec l'articulation des notes. Modifiez ensuite la ligne 4205 pour pouvoir entendre simultanément les différentes voix.

S'il le faut, ajoutez la voix médiane de la même manière, en l'exécutant d'abord séparément, puis avec la voix supérieure, et finalement avec les deux autres voix. Tout cela peut être réalisé en modifiant l'instruction 4205.

## LE JUGEMENT MUSICAL

Espérons que les considérations qui précèdent vous auront clairement montré combien il serait difficile de faire jouer "musicalement" à un ordinateur une pièce de musique classique. Plus on cherche à obtenir un effet agréable, plus on comprend à quel point l'exécutant s'éloigne de la partition écrite, dans l'intérêt d'un jugement personnel. Cependant, la pratique d'un musicien l'amène à un niveau où il n'a pas besoin de penser consciemment à l'altération de la durée d'une note, car il reconnaît intuitivement si "le son est bon". Nous devons par contre trouver et corriger chaque note avec une chaîne introduite par une instruction DATA !

Certaines personnes peuvent critiquer le programme précédent en considérant que le son est "électronique" et l'effet mécanique, malgré nos efforts pour l'humaniser. Il est exact que l'effet musical ne sera jamais très réaliste. Mais sa valeur est justement liée à notre appréciation du réalisme, à nos efforts pour reconnaître ce que l'exécutant humain offre "en plus", afin de pouvoir nous approcher un peu de ce qu'il accomplit. Notre appréciation de la musique et du talent d'un musicien ne peut qu'être augmentée par un tel exercice.


Allegretto
f
p
f
mf
p
p
p
f
poco rit.

# 10

# EFFETS SPÉCIAUX

Jusqu'à présent, nous n'avons pratiquement considéré le synthétiseur sonore que d'un point de vue purement musical. Comment l'utiliser pour agrémenter des jeux ?

## LE CANAL DE BRUIT

Il existe un paramètre supplémentaire de la commande SOUND que nous n'avons pas encore examiné dans cet ouvrage. Une période de bruit peut être ajoutée au son, et sa valeur peut varier de 1 à 31. Entrez les lignes suivantes pour écouter l'effet produit :

```
10 FOR J=1 TO 31
20 SOUND 1,0,100,4,0,0,J
30 NEXT J
```

et vous verrez que le son devient plus "profond" à mesure que s'élève la valeur de la période de bruit. Essayez :

```
 5 FOR J=1 TO 2
10 FOR J1=16 TO 31
20 SOUND 1,0,5+J1,5,0,0,J1
30 NEXT J1
40 FOR J1=31 TO 12 STEP -1
50 SOUND 1,0,7,6,0,0,J1
60 NEXT J1
70 NEXT J
```

et vous entendrez les rafales du vent qui souffle en tempête...

## BRUITS DE PAS, EXPLOSIONS, SCIES...

L'exemple précédent utilisait uniquement le canal de bruit. Il peut néanmoins être combiné avec des tons "normaux". Essayez :

```
10 FOR VOL=1 TO 7 STEP 0.25
20 SOUND 1,95,1,VOL,0,0,1
30 SOUND 1,400,1,VOL,0,0,31
40 SOUND 1,0,20,0
50 NEXT VOL
```

Cet exemple simple peut subir bien des modifications et être utilisé dans de nombreux jeux afin de produire des bruits de pas, de bombe à retardement, d'horloge, etc. (Dans certains cas, il vous faudra beaucoup d'imagination.)

Le son suivant évoque le bruit d'une scie circulaire entamant un tronc coriace :

```
10 ENT 1,10,1,8
15 ENV 1,5,1,5,1,0,200
20 SOUND 1,50,200,6,1,1,3
30 SOUND 1,48,50,5,1,1,6
35 SOUND 1,48,10,5,1,1,1
40 SOUND 1,50,150,6,1,1,3
```

Vous pouvez baptiser vous-même les bruits horribles que voici, en espérant que vous pourrez leur trouver une utilisation...

```
10 FOR J=1 TO 5
20 ENV 1,1,1,20
30 ENT -1,1,3,1
40 SOUND 1,284,-5,10,1,1,3
50 NEXT J
```

et

```
10 ENT -1,1,4,2,1,-4,2
20 SOUND 1,50,100,4,0,1
```

C'est un excellent domaine d'expérimentation. Il y a peu de règles,

mais une bonne compréhension de la syntaxe des diverses commandes sonores vous aidera à découvrir de nombreux sons fascinants.

## L'ATTENTE ET LE "FLUSH"

Lorsqu'on veut ajouter des sons à un jeu, la généreuse commande SOUND peut nous offrir une astuce dont nous n'avons pas encore parlé. Vous vous souvenez que chaque canal possède une queue sonore. Si l'on crée un son et qu'on lui associe l'indicateur FLUSH (en validant le bit 7), tous les sons qui se trouvent dans la queue — et même celui qui est joué à ce moment — sont instantanément annulés et le son associé au FLUSH est produit par le canal correspondant. De cette manière, si vous synchronisez une explosion avec une action particulière se déroulant sur l'écran, vous pouvez lier directement ce son à la frappe d'une touche déterminée.

Comme le FLUSH est défini par le bit 7, vous devrez avoir une instruction de ce genre :

```
  10 FLUSH=128
  ...
1010 SOUND 1+FLUSH,...
```

Le bit 6 — de valeur 64 — est appelé bit d'attente ; s'il est validé, il oblige le premier son (et tous ceux qui le suivent dans la queue) à demeurer en attente jusqu'au moment où il sera libéré. La commande de libération est la commande RELEASE ; ainsi :

```
RELEASE 4
```

libérera tous les sons en attente dans le canal C, sans affecter les autres canaux. Vous pouvez ainsi préparer les queues sonores puis, grâce à une simple commande RELEASE, libérer toutes les forces de l'enfer...

Si aucun son n'est en attente dans un canal au moment où une commande RELEASE est exécutée, cela n'aura aucune conséquence fâcheuse.

III

# LES POSSIBILITÉS GRAPHIQUES

Cette partie revient aux délices graphiques de l'Amstrad et explore certaines possibilités concernant la manipulation des couleurs et l'animation, en proposant également des exemples d'utilisation du graphisme pour les schémas et les diagrammes.

# 11

# ANIMATION ET ILLUSION

Avec une animation très simplifiée, nous voulons déplacer un personnage sur l'écran. Pour cela, il nous faudra afficher le personnage, puis l'effacer et l'afficher à une position voisine.

```
 5 PERSON$=CHR$(250)
10 CLS
20 Y=23 : LOCATE 1,Y : PRINT PERSON$;
30 FOR X=1 TO 39
40 LOCATE X,Y
50 PRINT " ";PERSON$;
60 NEXT
```

Vous avez pu remarquer que le personnage se déplace d'une manière un peu saccadée, avec des à-coups. Si vous insérez :

```
45 CALL &BD19
```

l'action sera nettement plus régulière. Cela est dû au fait que vous utilisez maintenant une routine (un sous-programme) du système d'exploitation qui attend le "retour de trame". C'est la période durant laquelle le système ne rafraîchit pas l'écran, et c'est donc le meilleur moment pour écrire sur l'écran.

Le retour de trame est directement lié au temps, et le résultat sera donc différent selon la position que vous donnerez au personnage. Modifiez la ligne 20 en :

```
20 Y=8 : ...
```

et vous comprendrez ce que cela signifie ! Des tests sont manifestement nécessaires.

## UNE BALLE QUI REBONDIT

En améliorant la technique simple exposée précédemment, nous pouvons écrire un programme permettant de faire rebondir une balle tout autour de l'écran. Chaque fois qu'elle touchera un bord, elle rebondira selon un angle particulier. Le programme de base est le suivant :

```
 10 CLS
 20 BALL$=CHR$(231)
 30 X=20 : Y=21
 40 XDIR= 1 : YDIR=1
 50 LOCATE X,Y : PRINT " ";
 60 X=X+XDIR : Y= Y+YDIR
 70 IF X<1 THEN X=1 : XDIR=1
 80 IF X>40 THEN X=40 : XDIR= -1
 90 IF Y<1 THEN Y=1 : YDIR=1
100 IF Y>25 THEN Y=25 : YDIR= -1
110 LOCATE X,Y
120 PRINT BALL$;
130 GOTO 50
```

Cela marche, mais on peut améliorer le programme. Ajoutez :

```
 15 BORDER 6
115 CALL &BD19
```

Sur certaines lignes, la balle est encore effacée avant même d'être affichée ! Essayons de la ralentir un peu.

```
125 FOR J=1 TO 5 : NEXT J
```

Voila qui est beaucoup mieux ! Si quelqu'un veut réaliser un jeu avec un citron qui rebondit, nous avons ce qu'il faut...

La ligne 125 représente une perte de temps. Il vaudrait mieux, dans un véritable jeu, considérer le programme précédent comme une sous-tâche que nous pourrions appeler à intervalles réguliers comme ceci :

```
EVERY 10 GOSUB
```

Quelques tests nous permettront de trouver la vitesse idéale.

Nous pourrions ajouter un peu de variété, des obstacles pour modifier la trajectoire de notre balle ? Pour l'instant, sa direction ne change que lorsqu'elle atteint le bord de l'écran. Pour "gérer" les obstacles, il nous faudra une méthode permettant de vérifier si la position de l'écran à laquelle nous voulons placer la balle est libre ou non. Cela n'est pas directement possible avec le BASIC (à moins de prendre en considération la commande TEST), mais il existe une routine du système d'exploitation intitulée TXT RD CHAR qui accomplit exactement ce que nous souhaitons en lisant et en donnant la valeur du caractère situé à la position actuelle du curseur. Les jeux des Chapitres 12 et 13 utilisent tous deux cette routine, et vous pouvez la copier dans vos propres programmes.

## JONGLER AVEC LES ÉCRANS ET LES ENCRES

La technique qui consiste à effacer et redessiner successivement une image est employée dans de nombreux jeux. Elle est aisée à programmer et se révèle assez efficace, mais elle a bien sûr des limites, car si vous voulez déplacer simultanément plus d'un caractère, il vous faudra effacer plusieurs emplacements avant de les redessiner, et vous aurez un programme plus lent.

Il existe d'autres moyens permettant de déplacer des objets sur l'écran. Dans le Chapitre 5, nous avons montré que nous pouvions conserver en RAM deux zones de mémoire écran et les faire alterner. L'exécution était quasi instantanée et pouvait donner l'impression d'un mouvement rapide. On dessine un objet à un emplacement donné en utilisant la mémoire écran normale, débutant en &C000, et on le redessine à un emplacement légèrement différent en utilisant la zone débutant à l'adresse &4000. Ensuite, il suffit de faire passer l'écran d'une zone à l'autre.

Cela nous procure un avantage : l'objet peut être bien plus large qu'un simple caractère. Mais il y a également un inconvénient : il faudra généralement plusieurs secondes pour dessiner le caractère,

et son déplacement sera lent. Les programmeurs avertis peuvent améliorer cette idée en exécutant un déroulement *(scrolling)* de la mémoire écran, de sorte que le caractère semble bouger, ou en exécutant des instructions POKE qui placent directement des données dans une mémoire écran tandis que l'autre est affichée. De telles méthodes sont communément utilisées dans les meilleurs jeux d'arcade, et elles sont programmées en langage d'assemblage.

A la fin du Chapitre 4, nous avons dit qu'un emploi judicieux des encres pouvait donner l'illusion que des objets apparaissaient et disparaissaient. Si nous sommes en mode 1, nous disposons de 4 couleurs. Une couleur est normalement assignée au fond de l'écran ; cela nous en laisse 3 pour dessiner. Si nous donnons à chaque encre la couleur du fond, nous pouvons alors dessiner sur l'écran 3 images différentes (non superposées), à raison d'une image par couleur de stylo, mais sans qu'aucune de ces images n'apparaisse. L'écran est apparemment vide. Ensuite, en modifiant successivement les encres, nous pouvons faire apparaître ou disparaître les objets à volonté, c'est-à-dire les déplacer !

Bien entendu, si nous utilisons le mode 0 et deux zones de mémoire écran, nous pourrons dessiner 15 objets dans chaque zone et montrer un objet à 30 positions différentes...

## PLANS MULTIPLES

Tout cela peut être très avantageux dans certaines circonstances, mais vous aurez remarqué que les images ne doivent pas se superposer, sans quoi l'image du "premier plan" effacerait l'image du "second plan". Comment créer l'illusion d'un objet émergeant de derrière un autre ?

Imaginons trois plans : un premier plan, un second plan, et un fond. Nous devons être capables de montrer :

- le fond (une simple couleur) ;
- un objet au second plan (cachant le fond) ;
- un objet au premier plan (cachant le fond) ;
- un objet au premier plan (cachant à la fois le second plan et le fond).

Cela fait 4 états différents. Et en mode 1 nous avons 4 encres différentes, ce qui devrait vous amener à réfléchir. Si nous utilisions l'encre 0 pour le fond, l'encre 1 pour le second plan, et les encres 2 et 3 pour les 2 états du premier plan ?

Cette idée semble prometteuse, mais comment pouvons-nous, lorsque nous écrivons au premier plan, décider rapidement s'il faut utiliser l'encre 2 ou 3 à un emplacement particulier ? Et comment effacer un objet du premier plan sans effacer également l'image du second plan qu'il pourrait cacher ?

Il devrait y avoir une réponse à cela. Et bien sûr il y en a une (sinon, nous ne nous serions pas donné la peine de faire tout ce développement !). La réponse réside dans le "mode d'encre graphique".

Nous n'en avons pas encore parlé. Quand vous écrivez un caractère ou dessinez une ligne en utilisant, par exemple, l'encre 1, vous pensez peut-être que tous les pixels nécessaires à sa composition ont la couleur de l'encre 1. Et ce peut effectivement être le cas si le mode d'encre graphique est défini comme mode normal, mais pas nécessairement s'il est dans un état différent ! Il possède 4 états, et dans 3 d'entre eux la nouvelle encre réagit d'une certaine manière avec l'ancienne.

*État 0* (Mode forcé) La nouvelle encre recouvre l'ancienne. C'est l'état normal, et celui que nous avons toujours utilisé jusqu'à présent.

*État 1* (Mode XOR — ou mode OU exclusif) La couleur du pixel est le résultat d'une opération de type OU exclusif entre la nouvelle encre et l'ancienne.

*État 2* (Mode AND — ou mode ET) Résultat d'une opération AND (et) entre la nouvelle couleur et l'ancienne.

*État 3* (Mode OR — ou mode OU) Résultat d'une opération OR (ou) entre la nouvelle couleur et l'ancienne.

(Pour vous remémorer la signification des expressions AND, OR et XOR, consultez le manuel de l'Amstrad, Chapitre 4).

Imaginons que nous avons mis le mode d'encre graphique à l'état 3 — en mode OR. Maintenant, si le fond est coloré par l'encre 0 et le second plan par l'encre 1, nous pouvons dessiner au premier plan

à l'encre 2. En fait, nous n'écrivons jamais avec l'encre 3. Le résultat sera :

| ANCIENNE ENCRE | NOUVELLE ENCRE | RÉSULTAT |
|---|---|---|
| 0 (fond) | 0 | 0 (fond) |
| 0 | 1 | 1 (second plan) |
| 0 | 2 | 2 (premier plan) |
| 1 (second plan) | 1 | 1 (second plan) |
| 1 | 2 | 3 (premier plan cachant le second plan) |

Bien entendu, les encres 2 et 3 doivent être de la même couleur pour que l'image paraisse toujours identique, mais en fait elles sont différentes car nous devons être capables de faire la différence lorsqu'il faut les effacer ! Néanmoins, en ce qui nous concerne, nous ''mettons'' dans notre stylo de l'encre 2, nous écrivons tranquillement, et les pixels déterminés seront colorés à l'encre 2 ou 3 selon la couleur initiale de leur emplacement.

En fait, nous avons sacrifié une des couleurs de notre palette pour avoir la possibilité d'écrire sur trois plans différents.

Comment effacer un objet du premier plan qui cache partiellement un objet du second plan ? Ou effacer un objet du second plan sans laisser de trace sur le fond ? Aucun problème. Il nous suffit de faire passer le mode d'encre graphique à l'état 2 (mode AND) et de sélectionner la couleur de stylo appropriée, 2 (pour effacer le second plan) ou 1 (pour effacer le premier plan). Supposons que nous voulions effacer un objet du second plan. Nous utilisons l'encre 2 avec notre stylo et les résultats sont les suivants :

| ANCIENNE ENCRE | NOUVELLE ENCRE | RÉSULTAT |
|---|---|---|
| 1 (second plan) | 2 | 0 (fond) |
| 3 (premier plan cachant le second plan) | 2 | 2 (premier plan) |

Ainsi, le premier plan reste apparemment intact (bien qu'en réalité il soit passé de l'encre 3 à l'encre 2). De la même manière, si

nous demeurons dans l'état 2 de l'encre graphique et que nous utilisons l'encre 1 avec notre stylo, nous pouvons effacer un objet du premier plan sans déranger le second plan :

| ANCIENNE ENCRE | NOUVELLE ENCRE | RÉSULTAT |
|---|---|---|
| 2 (premier plan) | 1 | 0 (fond) |
| 3 (premier plan cachant le second plan) | 1 | 1 (second plan) |

Le programme suivant est utile pour vérifier ce que nous venons de voir. Remarquez que la transparence est également validée.

```
10 MODE 1
20 INK 0,1:INK 1,24:INK 2,20:INK 3,6:PAPER 0:PEN 1
25 PRINT CHR$(22)+CHR$(1);: REM TRANSPARENCE
30 INK 1,6:INK 2,24
35 FOREPEN=2:MIDPEN=1:FORERUB=1:MIDRUB=2
40 PRINT CHR$(23)+CHR$(3);: REM SELECTION ENCRE GRAPHIQUE ETAT 3
   (OU) POUR DESSIN DU FOND OU IMAGES SECOND PLAN OU TEXTE.
50 PENCIL=FOREPEN:GOSUB 1000:REM DESSIN FOND
60 PENCIL=MIDPEN:GOSUB 2000:REM DESSIN SECOND PLAN
65 LOCATE 10,25:PRINT "Pressez une touche ";
70 Z$=INKEY$:IF Z$="" THEN 70
80 PRINT CHR$(23)+CHR$(2);: REM SELECTION ENCRE GRAPHIQUE ETAT 2
   (AND) POUR EFFACER FOND OU IMAGES SECOND PLAN OU TEXTE.
90 PENCIL=MIDRUB:GOSUB 2000: REM EFFACEMENT IMAGE SECOND PLAN
100 PRINT CHR$(22)+CHR$(0);: REM SUPPRIMER LA TRANSPARENCE
110 PRINT CHR$(23)+CHR$(0);: REM SELECTION ENCRE GRAPHIQUE
    ETAT NORMAL (FORCE)
120 END
1000 REM AFFICHAGE OU EFFACEMENT TEXTE
1010 LOCATE 20,12
1020 PEN PENCIL
1030 PRINT "       Ce texte est affiché sur le fond";
1999 RETURN
2000 REM AFFICHAGE OU EFFACEMENT TRIANGLE
2010 MOVE 100,100
2020 DRAW 540,100,PENCIL
2030 FOR J=100 TO 540 STEP 2
2040 MOVE 320,360
2050 DRAW J,100,PENCIL
2060 NEXT J
2999 RETURN
```

Cette méthode est très puissante et vaut largement la peine d'être explorée !

## ON SE MONTRE ET ON SE CACHE !

Le même genre de trucage est utilisé dans le programme suivant, qui nous montre comment deux images peuvent être dessinées au

même endroit, puis affichées alternativement pour donner l'impression de disparaître et de réapparaître :

```
10 MODE 1 : DEG
20 GOSUB 1000 : REM DESSIN IMAGE 1
30 GOSUB 2000 : REM DESSIN IMAGE 2
40 GOSUB 6000 : REM DESSIN LEGENDE
50 FOR SCREEN=1 TO 10
60   GOSUB 3000 : REM AFFICHAGE IMAGE 1
70   GOSUB 4000 : REM RETARD
80   GOSUB 5000 : REM AFFICHAGE IMAGE 2
90   GOSUB 4000 : REM RETARD
100 NEXT SCREEN
110 INK 0,1
120 INK 1,24
130 INK 2,20
140 INK 3,6
150 PAPER 0
160 PEN 1
170 PRINT CHR$(23)+CHR$(0);
180 END
1000 REM DESSIN IMAGE 1
1010 PRINT CHR$(23)+CHR$(3);
1020 FOR Z=1 TO 360
1030     PLOT 320,200,1
1040     DRAW 320+90*COS(Z),200+90*SIN(Z),1
1050 NEXT z
1999 RETURN
2000 REM DESSIN IMAGE 2
2010 PRINT CHR$(23)+CHR$(3);
2020 PLOT 100,100,2
2030 DRAW 540,100,2
2040 DRAW 320,360,2
2050 DRAW 100,100,2
2060 MOVE 320,360,2
2999 RETURN
3000 REM AFFICHAGE IMAGE 1
3010 INK 0,2
3020 INK 1,1
3030 INK 2,2
3040 INK 3,1
3999 RETURN
4000 REM RETARD
4010 TM=TIME+300
4020 WHILE TIME<TM
4030 WEND
4999 RETURN
5000 REM AFFICHAGE IMAGE 2
5010 INK 0,4
5020 INK 1,4
5030 INK 2,0
5040 INK 3,0
5999 RETURN
6000 REM AFFICHAGE LEGENDE
6010 LOCATE 18,24
6020 PRINT CHR$(23)+CHR$(0);
6030 PEN 3
6040 PRINT "Ceci est une légende";
6999 RETURN
```

Remarquez que le programme précédent modifie le fond ; mais ce n'est pas nécessaire, et dans la plupart des cas l'illusion sera plus grande si vous ne changez pas la couleur du fond.

Remarquez aussi que le sous-titre est affiché avec les deux images.

Les techniques précédentes ne sont que de petits exemples de tout ce qui peut être réalisé. Il est essentiel de comprendre ces méthodes si vous voulez travailler sur des dessins en trois dimensions, des solides en rotation, des dessins proportionnels, etc.

## FUTURS HORIZONS

Avant d'abandonner ce sujet, considérons un moment le fait que nous avons utilisé le mode 1. Comment faire en mode 0, qui nous permet d'employer 16 encres ? Et si, au lieu de 3 plans, nous en avions 4 ? Il nous faudrait :

- fond ;
- troisième plan ;
- deuxième plan ;
- deuxième plan cachant le troisième plan ;
- premier plan ;
- premier plan cachant le troisième plan ;
- premier plan cachant les deuxième et troisième plans.

Cela réquisitionnerait 7 de nos encres. Nous pourrions utiliser les autres pour différentes couleurs de premier plan, ou de deuxième plan, ou pour faire apparaître et disparaître des objets du premier plan, etc. Il vous faudra un crayon et du papier pour calculer les effets des opérations OU sur les diverses encres. Faites une colonne pour l'ancienne encre, une pour la nouvelle encre, et une pour le résultat... Vous verrez que de nombreuses combinaisons sont possibles.

# 12

# JEU N° 1 – LE HEFFALUMP AFFAMÉ

## JEUX SUR ORDINATEUR – ARCANE OU ARCADE ?

Nous ne nous excusons pas de présenter des jeux ! Bien conçus, ils peuvent être très amusants et éducatifs. Les talents de programmation nécessaires pour réaliser un jeu attrayant sont parmi les plus dignes, et bien des programmeurs les ont précisément acquis parce qu'ils étaient amenés à créer des jeux de plus en plus intéressants.

Les personnes qui dénigrent les jeux d'arcade ne considèrent généralement que le résultat final (souvent limité, il est vrai) et ne voient pas les efforts qui ont précédé leur création. La plupart des programmeurs passent très peu de temps à jouer avec le jeu qui est le produit de leur travail ; ils sont trop occupés à concevoir leur prochaine œuvre.

## AVEC TOUTES NOS EXCUSES à A. A. MILNE

Notre premier jeu présente un heffalump simple d'esprit dont la tanière se trouve dans le coin nord-est d'un bois. Il est inutile de vous rappeler que le heffalump a une mauvaise vue et qu'il fait surtout confiance à son odorat lorsqu'il chasse une proie.

Dans le cas qui nous occupe, les victimes sont les membres d'un groupe d'explorateurs qui tentent, un par un, de traverser le bois en partant du coin nord-ouest pour atteindre le coin sud-est, où se trouve une rivière. L'aversion du heffalump pour l'eau est bien connue, et s'ils parviennent à plonger dans la rivière ils seront saufs et pourront siffloter un petit air de victoire méprisant pour exprimer leur soulagement.

Le joueur contrôle le déplacement des explorateurs grâce aux touches flèches. Lorsqu'ils sont envoyés dans une direction, les explorateurs courent droit devant eux dans une panique aveugle, rebondissant sur les arbres et regardant certainement d'un air effrayé par-

dessus leur épaule pour voir où se trouve le monstre maladroit. Apparemment, ils ne cherchent pas la difficulté ; ils se déplacent presque toujours en ligne droite et paraissent apprécier les angles droits.

Le monstre, qui se guide à l'odeur, a également tendance à se cogner aux arbres, mais comme il possède un cerveau encore plus petit que ses victimes potentielles, il ne cherche pas à contourner l'obstacle et pousse vainement dans la direction où il croit devoir aller. Cependant, à mesure que le jeu avance, sa vitesse augmente de plus en plus, soit parce qu'il est furieux de voir ses victimes s'échapper, soit parce que son goût du sang devient plus vif. Le dernier explorateur qui tente la traversée a bien peu de chances de s'échapper, car la bête est prise d'une véritable frénésie sanguinaire...

Vous avez dû maintenant vous rendre compte de la profondeur intellectuelle de ce jeu, et nous allons donc entamer sa programmation.

## COMMENTAIRE SUR LE PROGRAMME

Le programme principal est assez court puisqu'il n'occupe que les lignes 5 à 170. Les lignes 10 à 40 installent la scène, les lignes 50 à 120 constituent le corps du programme et les lignes 130 à 170 forment la conclusion.

```
  5 REM HEFFALUMP AFFAME COPYRIGHT (C) 1984 JOHN BRAGA
 10 GOSUB 1000      : REM INITIALISATION
 20 GOSUB 2000      : REM DESSIN DE LA BORDURE
 30 GOSUB 2500      : REM DESSIN DE LA FORET
 40 GOSUB 3000      : REM PLACER LE HEFFA
 50 FOR EXPLO=1 TO NOMBREXPLO
 60      GOSUB 3400 : REM DEPART EXPLORATEUR
 70      GOSUB 3800 : REM DEPART HEFFA
 80      GOSUB 4000 : REM DEPLACEMENT EXPLORATEUR
 90      GOSUB 5000 : REM RETOUR POSITION DE DEPART
120 NEXT EXPLO
130 GOSUB 6000 : REM MESSAGE FINAL
140 CLS
150 PEN 1
160 SPEED KEY 20,3
170 END
```

Si vous le désirez, la conclusion peut restaurer les couleurs originales.

```
1000 REM INITIALISATION
1010 EXPLO$=CHR$(249) : BORD$=CHR$(143)
1020 NOMBREXPLO=10
1040 ARBRE$=CHR$(229):NOMBREDARBRES=40
1050 RIVIERE$=CHR$(207):HEFFA$=CHR$(184)
1060 LIGNEHAUT=2:LIGNEBAS=25:LIGNEGAUCHE=1:
     LIGNEDROITE=40
1070 HEFFAINTERVALLE=5
1080 VRAI= -1:FAUX=0
1090 ECHAPPE=0:CAPTURE=0
1100 MODE 1
1110 INK 0,9
1120 PAPER 0
1130 INK 1,24
1140 PEN 1
1150 INK 2,20
1160 INK 3,6
1170 SPEED KEY 4,2
1200 MEMORY &AB77
1210 RESTORE 1250
1220 FOR J=1 TO 8
1230 READ X:POKE HIMEM+J,X:NEXT J
1240 DATA &CD,&60,&BB,&32,&7F,&AB,&C9,0
1250 RDCHAR=&AB78 :CHREAD=&AB7F
1999 RETURN
```

En 1000 se trouve le sous-programme d'initialisation. Comme il n'est exécuté qu'une fois, il serait plus correct de l'inclure dans le programme principal, mais si l'on veut être méthodique, il vaut mieux placer ensemble toutes les lignes d'initialisation.

Vous voudrez certainement tester de nombreuses variables de ce jeu, et ajouter vos propres améliorations. Il est possible de modifier la valeur de NOMBREXPLO (le nombre des explorateurs) à la ligne 1020.

La bordure du jeu est installée par la ligne 1060, mais vous pouvez jouer en mode 2, sur 80 colonnes, en modifiant cette ligne et la ligne 1100. Bien entendu, il faudra alors changer les instructions INK et

PEN en tenant compte du fait que vous ne disposerez plus que de deux couleurs.

La ligne 1070 nous aide à contrôler la rapidité des réactions de l'heffa. Modifiez-la en fonction de vos préférences et de votre dextérité. Ceux qui n'aiment pas perdre devraient donner à HEFFAINTERVALLE une valeur minimale de 10... Voyez aussi la ligne 3805.

L'instruction SPEED KEY utilise la caractéristique très utile de l'Amstrad permettant de définir la vitesse à laquelle les touches réagissent à votre frappe. Le seul ennui, avec cette méthode, est que si vous exécutez le programme et que vous faites une erreur de frappe, vous vous rendrez compte que le clavier devient pratiquement incontrôlable ! Pour éviter ce problème, entrez les instructions :

```
  2 ON ERROR GOTO 200
  3 ON BREAK GOSUB 150
200 PRINT "ERREUR";ERR;"A LA LIGNE";ERL
210 GOTO 150
```

Le sous-programme d'affichage de la bordure est très simple et ne requiert aucun commentaire :

```
2000 REM DESSIN DU BORD
2010 FOR J=LIGNEGAUCHE TO LIGNEDROITE
2020      LOCATE J,LIGNEHAUT : PRINT BORD$;
2030      LOCATE J,LIGNEBAS : PRINT BORD$;
2040 NEXT J
2050 FOR J=LIGNEHAUT TO LIGNEBAS
2060      LOCATE LIGNEGAUCHE,J : PRINT BORD$;
2070      LOCATE LIGNEDROITE,J : PRINT BORD$;
2080 NEXT J
2100 RIVIEREXPOS=LIGNEDROITE:RIVIEREYPOS=LIGNEBAS-3
2110 LOCATE RIVIEREXPOS,RIVIEREYPOS: PRINT RIVIERE$;
2499 RETURN
```

Vient ensuite le sous-programme qui place les arbres. Vous pourrez modifier le nombre d'arbres affiché.

```
2500 REM DESSIN DE LA FORET
2510 RANDOMIZE TIME
2520 FOR J=1 TO NOMBREDARBRES
```

```
2530 X% = RND(1)*LIGNEDROITE + LIGNEGAUCHE : IF
      X% > = LIGNEDROITE OR X% < = LIGNEGAUCHE THEN 2530
2540 Y% = RND(1)*LIGNEBAS + LIGNEHAUT : IF
      Y% > = LIGNEBAS OR Y% < = LIGNEHAUT THEN 2540
2550 LOCATE X%,Y% : PRINT ARBRE$;
2560 NEXT J
2570 LOCATE RIVIEREXPOS - 1,RIVIEREYPOS : PRINT " ";
2999 RETURN
```

L'instruction RANDOMIZE nous permet d'afficher chaque fois une forêt différente.

Remarquez la ligne 2570 qui s'assure que les victimes disposent bien d'un chemin libre pour s'échapper !

```
3000 REM PLACER LE HEFFA
3010 HEFFXPOS = LIGNEDROITE - 1 : HEFFYPOS = LIGNEHAUT + 1
3015 PEN 3
3020 LOCATE HEFFXPOS,HEFFYPOS : PRINT HEFFA$;
3399 RETURN
```

Un heffa est dessiné en rouge vif dans le coin supérieur droit de l'écran.

```
3400 REM DEPART EXPLORATEUR
3405 PEN 2
3410 LOCATE LIGNEGAUCHE + 1,LIGNEHAUT + 1:PRINT EXPLO$;
3420 EXPLOXPOS = LIGNEGAUCHE + 1:EXPLOYPOS = LIGNEHAUT + 1
3430 CHASSE = VRAI:TROUVE = FAUX
3799 RETURN
```

Remarquez que CHASSE et TROUVE sont considérées comme des variables booléennes, assez familières à ceux qui programment en Pascal.

```
3800 REM DEPART HEFFA
3805 J = HEFFAINTERVALLE + (NOMBREXPLO - EXPLO)*2
3810 EVERY J GOSUB 7000
3999 RETURN
```

```
7000 REM TACHE SEPAREE POUR DEPLACER LE HEFFA VERS
       L'EXPLORATEUR
7005 X=HEFFXPOS:Y=HEFFYPOS
7010 IF HEFFXPOS > EXPLOXPOS THEN X=HEFFXPOS-1
       ELSE IF HEFFXPOS < EXPLOXPOS THEN X=HEFFXPOS+1
7020 IF HEFFYPOS > EXPLOYPOS THEN Y=HEFFYPOS-1
       ELSE IF HEFFYPOS < EXPLOYPOS THEN Y=HEFFYPOS+1
7025 LOCATE X,Y:CALL RDCHAR
7027 IF PEEK(CHREAD)=32 THEN 7030
7028 IF PEEK(CHREAD)ASC(EXPLO$) THEN CHASSE=FAUX:
       TROUVE=VRAI:CAPTURE=CAPTURE+1 ELSE 7999
7030 LOCATE HEFFXPOS,HEFFYPOS: PRINT " ";: PEN 3:
       LOCATE X,Y:PRINT HEFFA$;: HEFFXPOS=X:HEFFYPOS=Y
7032 CALL &BD19
7999 RETURN
```

Le déplacement du heffa est effectué par une sous-tâche séparée qui est initialisée par le sous-programme 3800. La ligne 3805 est cruciale pour déterminer la rapidité des réactions du heffa, et vous pouvez ralentir le monstre en trichant un peu.

```
4000 REM MOUVEMENT DE L'EXPLORATEUR
4005 IF NOT CHASSE THEN 4999
4007 DI
4010 Z$=INKEY$ : IF Z$=" " THEN 4800
4020 Z=ASC(Z$)
4030 IF Z<240 OR Z>243 THEN 4010
4040 Z=Z-239
4050 ON Z GOSUB 4100,4200,4300,4400
4100 REM HAUT
4110 XDIR=0 : YDIR= -1
4120 GOTO 4800
4200 REM BAS
4210 XDIR=0 : YDIR=1
4220 GOTO 4800
4300 REM GAUCHE
4310 YDIR=0 : XDIR= -1
4320 GOTO 4800
4400 REM DROITE
4410 YDIR=0 : XDIR=1
```

```
4800 X=EXPLOXPOS+XDIR : Y=EXPLOYPOS+YDIR
4805 LOCATE X,Y: CALL RDCHAR : CH$=CHR$(PEEK(CHREAD)):
        IF CH$=RIVIERE$ THEN CHASSE=FAUX : GOTO 4900
4810 IF CH$=HEFFA$ THEN CHASSE=FAUX:TROUVE=VRAI:
        CAPTURE=CAPTURE+1:LOCATE EXPLOXPOS,EXPLOYPOS:
        PRINT " ";: GOTO 4930
4820 IF CH$<>" " THEN XDIR=0-XDIR : YDIR=0-YDIR:
        GOTO 4930
4900 LOCATE EXPLOXPOS,EXPLOYPOS:PRINT " ";: LOCATE X,Y:
        PEN 2: PRINT EXPLO$;:EXPLOXPOS=X: EXPLOYPOS=Y
4930 EI
4940 GOTO 4005
4999 RETURN
```

Le sous-programme 4000 constitue la boucle interne du programme principal. L'ordinateur exécute ce sous-programme jusqu'au moment où l'explorateur est capturé ou parvient à s'échapper, mais bien entendu le sous-programme est régulièrement interrompu par la sous-tâche qui déplace le heffa.

Si une touche est pressée, elle est analysée. Seules les touches flèches sont valables, et toutes les autres touches sont ignorées. De cette manière, il est assez facile d'adapter ce jeu pour l'utiliser avec un *joystick.*

Les lignes 4100 à 4410 déterminent la direction. Une fois déterminée, elle demeure inchangée. Vous pouvez modifier le programme pour permettre à l'explorateur de tourner à 45 degrés plutôt qu'à 90 degrés. Cela augmenterait son agilité lorsqu'il est sur le point d'être capturé.

La ligne 4800 place la position souhaitée dans les variables X et Y. La position actuelle est maintenue à EXPLOXPOS, EXPLOYPOS. Ensuite, on appelle un sous-programme en langage d'assemblage qui a été chargé durant la phase d'initialisation. Ce sous-programme lit le caractère qui se trouve à la position souhaitée pour éviter les collisions. Si le caractère est RIVIERE$ ou HEFFA$, la chasse est terminée — du moins pour ce joueur. Si le caractère n'est pas un espace, il doit s'agir d'un arbre ou de la bordure, et la direction est alors inversée.

Remarquez les instructions DI et EI, qui sont essentielles si vous ne voulez pas que des explorateurs rouges apparaissent n'importe où. (Cela peut arriver si l'interruption de la routine de déplacement du heffa se produit après que vous ayez placé le curseur en

EXPLOXPOS,EXPLOYPOS, mais avant que vous ayez pu afficher l'explorateur en bleu ; quand vous retournez dans le sous-programme, la position a été modifiée et le stylo est rouge !)

Le sous-programme 5000 remet tout en ordre.

```
5000 REM REMISE EN ORDRE
5005 DI
5007 PEN 1
5010 LOCATE RIVIEREXPOS,RIVIEREYPOS : PRINT RIVIERE$;
5015 EXPLOXPOS=LIGNEDROITE-1:EXPLOYPOS=LIGNEHAUT+1
5017 LOCATE 1,LIGNEHAUT-1 : PRINT "ECHAPPES:";
     EXPLO-CAPTURE;".   CAPTURES:";CAPTURE
5020 EI
5030 IF CAPTURE THEN GOSUB 10000 ELSE GOSUB 9000
5999 RETURN
```

La rivière est replacée au bon endroit, pour plus de sûreté. Une nouvelle position est préparée pour afficher le nouvel explorateur, s'il en reste. Ensuite, le score est affiché sur la ligne supérieure. Enfin, le programme choisit la mélodie adaptée au résultat.

```
 9000 REM CHANT DE VICTOIRE !
 9010 RSTORE 9800
 9020 READ T
 9030 IF T=-1 THEN 9900
 9040 SOUND 1,T,20,5
 9050 GOTO 9020
 9800 DATA 80,60,47,80,60,47,80,60,47
 9810 DATA -1
 9900 WHILE SQ(1)>127 : WEND
 9999 RETURN
10000 REM CHANT FUNEBRE !
10010 RESTORE 10800
10020 READ T,L
10030 IF T=-1 THEN 10900
10040 SOUND 1,T,L*40,5
10045 SOUND 1,0,5,0
10050 GOTO 10020
10800 DATA 478,2,478,1.5,478,.5,478,2,402,1.5,426,.5
           426,1.5,478,.5,478,1.5,536,.5,478,2
```

```
10810 DATA -1,-1
10900 WHILE SQ(1)>127:WEND
10999 RETURN

 6000 REM MESSAGE FINAL
 6010 J=REMAIN(0)
 6999 RETURN
```

Ce sous-programme met fin aux activités du heffalump en supprimant la sous-tâche. A vous de programmer, si vous le voulez, le message final...

# 13

# JEU N° 2 — FANTÔMES EN BOUTEILLE !

*Note de l'auteur :* Je ne peux pas m'attribuer la conception de ce jeu, qui a été imaginé par mon fils, âgé de dix ans. Je n'ai pas cherché à savoir d'où il avait pu tirer une telle idée.

Le programme et le commentaire sont donnés dans ce chapitre, pour que vous puissiez suivre la logique du jeu et (qui sait ?) l'améliorer encore.

## SCÉNARIO

Le ''scénario'' est simple. Un héros (vous, bien sûr) voit le jour dans le coin supérieur droit d'un écran assez désertique. Les seuls objets qui s'y trouvent sont une bouteille (dans le coin inférieur gauche) et une tombe (en bas à droite).

Soudain, un fantôme à l'air mauvais émerge de la bouteille en poussant un gémissement, bientôt suivi à intervalles réguliers par plusieurs de ses semblables. Ils paraissent décidés à faire votre connaissance et se dirigent vers vous d'une manière assez déconcertante en formant une hideuse caravane.

Votre unique chance de survie consiste :

- à reboucher la bouteille avant qu'un trop grand nombre de fantômes se soient échappés ;
- en jouant le rôle d'appât, à les entraîner dans la tombe (en passant par la porte supérieure), à vous échapper par la porte inférieure, que vous devrez refermer derrière vous avec une pierre, puis à faire le tour de la tombe pour les enfermer en bloquant la porte supérieure avec une autre pierre.

Si vous échouez, vous êtes voué à fuir éternellement, pourchassé par les fantômes. Une autre solution, tout aussi déplaisante, consiste

à vous enfermer involontairement dans la tombe, où vous demeurerez à tout jamais tandis que les fantômes continueront leurs gémissements à l'extérieur.

*Note de l'auteur :* Inutile de m'écrire pour me faire remarquer que des fantômes qui se respectent ne sont pas arrêtés par une simple pierre. L'idée n'est pas de moi, je me suis contenté de concevoir le programme...

## COMMENT JOUER ?

Quand vous aurez chargé le programme de la manière habituelle, utilisez les quatre touches flèches pour vous déplacer sur l'écran, et la touche COPY pour bloquer l'entrée et la sortie de la tombe. Cette dernière touche peut être pressée lorsque vous vous trouvez exactement devant l'entrée ou la sortie — sinon, elle ne produit pas le moindre effet.

Notez bien que vous devez absolument faire entrer les fantômes par l'issue supérieure, et donc bloquer d'abord l'issue inférieure ! Essayez le contraire et vous verrez ce que nous voulons dire...

Pour boucher la bouteille (la première chose à faire si vous possédez la moindre parcelle de bon sens), il vous suffit de vous placer devant le goulot. Il n'est pas nécessaire de presser une touche.

## COMMENTAIRE SUR LE PROGRAMME

Le programme utilise diverses techniques semblables à celles que nous avons employées dans le premier jeu, mais il y a deux sous-tâches indépendantes et une routine en assembleur beaucoup plus longue. Elle sert à contrôler la vitesse. Si l'on utilisait uniquement le BASIC, il serait un peu long de déterminer les mouvements des dix spectres. L'emploi judicieux de l'assembleur nous fournit une réponse satisfaisante à ce problème.

Nous étudierons le moyen de créer un sous-programme en assembleur dans le Chapitre 16.

Les lignes 7 à 50 s'occupent d'installer le décor. Des instructions GOSUB envoient en 1000 (initialisation), en 2000 (dessin de la bouteille et de la tombe), en 3000 (affichage du héros) ; une sous-tâche

séparée (4000) est exécutée toutes les 2 500 ms et fera chaque fois émerger un fantôme, jusqu'à un maximum de dix ou bien jusqu'à ce que vous ayez réussi à boucher la bouteille.

```
 7 REM COPYRIGHT (C) 1984 JOHN BRAGA
10 REM FANTOMES EN BOUTEILLE!
20 GOSUB 1000
30 GOSUB 2000
40 GOSUB 3000
50 EVERY 50 GOSUB 4000
```

Le jeu proprement dit est exécuté durant un unique appel au sous-programme 5000, lancé par la ligne 60. Il déplace simplement le héros, car l'apparition des fantômes et leurs mouvements sont contrôlés par des sous-tâches séparées ; c'est une méthode simple (avec cette version du BASIC !) et qui vous décharge des problèmes de synchronisation.

```
60 GOSUB 5000
```

Les routines finales se trouvent aux lignes 70 à 300. Les mouvements des spectres (ou leurs apparitions) sont interrompus et le programme calcule leurs positions pour voir s'ils se trouvent à l'intérieur ou à l'extérieur de la tombe. S'ils se trouvent tous à l'intérieur, un message de remerciement est affiché, mais si certains sont encore à l'extérieur, c'est un message de condamnation. Si un fantôme vous attrape, le message "VOUS ETES PRIS!" s'affichera.

```
 70 X=REMAIN(0):Y=REMAIN(1)
 80 IF NOT FREE THEN 200
100 FOR J=1 TO NUMSPECTRE
110 GX=PEEK(GXSTORE+2*J-2):GY=PEEK(GXSTORE+2*J-1)
120 IF GX<31 OR GX>33 OR GY<16 OR GY>20
      THEN DOOMED=TRUE
130 NEXT J
140 IF DOOMED THEN LOCATE 15,1:PRINT "VOUS ETES
      CONDAMNE A FUIR ETERNELLEMENT!!!":GOTO 210
150 LOCATE 20,1: PRINT "L'HUMANITE ENTIERE VOUS REMERCIE!"
160 GOTO 210
200 LOCATE 20,1 : PRINT "VOUS ETES PRIS!!!!!";
```

```
210 FOR J=1 TO 4000: NEXT J
220 CLS
240 SPEED KEY 10,10
250 PEN 1
300 END
```

Le sous-programme d'initialisation débutant en 1000 réserve d'abord une partie de la mémoire pour la routine en assembleur, et la charge ensuite à partir de l'adresse 7000h. Le chargement s'effectue en lisant les instructions DATA et en exécutant des instructions POKE plutôt qu'en chargeant un fichier binaire conservé sur cassette. C'est un petit avantage supplémentaire.

De plus, au lieu de taper "manuellement" toutes les instructions DATA, ce qui est une tâche assez longue et qui peut entraîner des erreurs, nous avons utilisé ZEN pour afficher en haut de l'écran le contenu de la zone où est stocké le code objet, puis nous sommes passés en BASIC et avons utilisé la commande COPY pour créer les instructions DATA l'une après l'autre, chacune contenant 8 octets. Une fois les données "capturées" de cette manière, nous avons corrigé les lignes pour ajouter les virgules et les "&" entre les valeurs des octets.

```
1000 REM INITIALISATION
1005 TXTRD=&7000: RDCHAR=&7007:MVGHOST=&7008:
      DONEFLAG=&70FE:NUMGHOSTS=&70FF:MXSTORE=
      &7100:GXSTORE=&7102
1010 CLS
1015 MEMORY &6FFF
1017 SPEED KEY 8,3
1020 BOTTLECOLOUR=1:MANCOLOUR=2:GHOSTCOLOUR=3
1030 MAN$=CHR$(249): GHOST$=CHR$(225):EDGE$=CHR$(143)
1040 TRUE= -1:FALSE=0
1050 STOPPERED=FALSE
1060 INK 3,6,5
1070 FREE=TRUE:PLAYING=TRUE: TOMBSTONES=0:
       DOOMED=FALSE
1080 STOPPER$=CHR$(210)
1085 ENT -1,10,10,3
```

```
1090 POKE NUMBGHOSTS,O: POKE DONEFLAG,FALSE
1100 GOSUB 7000: CLS
1500 RESTORE 1900
1510 FOR J=0 TO 2000
1520 READ X:IF X= -1 THEN 1590
1530 POKE &7000+J,X
1580 NEXT J
1590 REM
1900 DATA &CD,&60,&BB,&32,&07,&70,&C9,0
1910 DATA &AF,&32,&FE,&70,&3A,&FF,&70,&32
1911 DATA &FD,&70,&2A,&00,&71,&E5,&C1,&21
1912 DATA &2,&71,&C5,&56,&23,&5E,&23,&E5
1913 DATA &D5,&79,&BA,&28,&06,&38,&3,&14
1914 DATA &18,1,&15,&78,&BB,&28,6,&38
1915 DATA 3,&1C,&18,1,&1D,&EB,&E5,&CD
1916 DATA &75,&BB,&CD,&60,&BB,&D1,&E1,&D5
1917 DATA &FE,&F9,&20,&A,&3E,1,&32,&FE
1918 DATA &70,&32,&FD,&70,&18,4,&FE,&20
1919 DATA &20,&1C,&CD,&75,&BB,&3E,&20,&CD
1920 DATA &5D,&BB,&E1,&E5,&CD,&75,&BB,&3E
1921 DATA &E1,&CD,&5D,&BB,&D1,&E1,&2B,&2B
1922 DATA &72,&23,&73,&23,&E5,&D5,&3A,&FD
1923 DATA &70,&3D,&32,&FD,&70,&D1,&E1,&C1
1924 DATA &20,&A0,&C9,-1
RETURN
```

Les instructions sont affichées par le sous-programme 7000.

```
7000 REM INSTRUCTIONS
7010 LOCATE 10,1:PRINT "FANTOMES EN BOUTEILLE!"
7020 PRINT
7030 PRINT "Vous êtes poursuivi par des fantômes "
7040 PRINT "qui s'échappent d'une bouteille."
7050 PRINT "Vous devez d'abord courir jusqu'à la "
7060 PRINT "bouteille (grâce aux touches fléchées)"
7070 PRINT "et la boucher en touchant le goulot. "
7080 PRINT "Cela empêche l'apparition d'autres "
7090 PRINT "fantômes. Vous devez alors entraîner les"
7100 PRINT "spectres gémissants vers la TOMBE et les"
7110 PRINT "enfermer pour toujours! N'oubliez pas de"
```

```
7120 PRINT "fermer d'abord l'issue INFERIEURE (quand"
7130 PRINT "les fantômes sont entrés). Si vous y "
7140 PRINT "parvenez, courez alors jusqu'à la porte"
7150 PRINT "SUPERIEURE et fermez-la derrière eux.
7160 PRINT
7170 PRINT "Pour fermer une issue, appuyez sur la "
7180 PRINT "touche COPY lorsque vous vous trouvez"
7190 PRINT "exactement devant la porte." :PRINT
7200 INPUT "Pressez ENTER pour commencer à jouer...",Z$
7999 RETURN
```

Le sous-programme permettant de dessiner la bouteille et la tombe ne mérite pas de commentaire, sauf pour préciser que si vous désirez placer ces objets à un autre endroit, vous devrez également effectuer des modifications dans le sous-programme qui vérifie si tous les fantômes sont enfermés dans la tombe, et dans le sous-programme qui vérifie si le héros a bouché la bouteille.

```
2000 REM DESSIN DE LA BOUTEILLE ET DE LA TOMBE
2010 LOCATE 1,21
2015 PEN BOTTLECOLOUR
2020 PRINT CHR$(209); EDGE$;CHR$(211)
2030 CHR$(214); EDGE$;CHR$(215)
2040 PRINT STRING$(3,EDGE$)
2050 PRINT STRING$(3,EDGE$)
2100 LOCATE 30,15
2110 PRINT EDGE$;EDGE$;" ";EDGE$;EDGE$
2120 FOR J=1 TO 5
2130 LOCATE 30,J+15:PRINT EDGE$;"   ";EDGE$
2140 NEXT J
2150 LOCATE 30,21:PRINT STRING$(3,EDGE$);" ";EDGE$
2999 RETURN
```

Le sous-programme 3000, "départ du héros", enregistre la position MX,MY et MXSTORE,MYSTORE pour la passer au sous-programme en assembleur. Ensuite, il interdit toute interruption afin de pouvoir tranquillement dessiner le héros sur l'écran (cette interdiction peut paraître superflue puisque les sous-tâches n'ont pas encore été créées. Disons qu'il s'agit d'une précaution en vue de modifications ultérieures!).

```
3000 REM DEPART DU HEROS
3010 MX=40 : MY=1
3020 NMX=MX : NMY=MY
3030 POKE MXSTORE,MX : POKE MXSTORE+1,MY
3040 DI : LOCATE MX,MY : PEN MANCOLOUR : PRINT MAN$; : EI
3999 RETURN
```

En 4000 se trouve une sous-tâche qui est appelée toutes les 2 500 ms ; elle vérifie si la bouteille est bouchée ou si les dix fantômes ont déjà été lâchés sur le pauvre monde. Si c'est le cas, la sous-tâche annule poliment toutes ses futures prestations grâce à une instruction REMAIN et abandonne le contrôle de l'exécution sans plus attendre.

S'il s'avère que son travail n'est pas terminé, elle libère un fantôme. Remarquez la technique simple qui consiste à dessiner une ligne, puis à la faire disparaître en la redessinant dans la couleur du fond.

```
4000 REM SORTIE D'UN FANTOME
4005 NUMSPECTRE=PEEK(NUMGHOSTS)
4010 IF STOPPERED OR (NUMSPECTRE=10) THEN J=REMAIN(0):
        GOTO 4999
4020 DI : PEN GHOSTCOLOUR : PLOT 20,82,GHOSTCOLOUR:
        DRAW 40,360,GHOSTCOLOUR : LOCATE 5,1 :
        PRINT GHOST$ :PLOT 20,82,0 : DRAW 40,360,0 : EI
4030 SOUND 1,0,20,7
4040 POKE NUMGHOSTS,NUMSPECTRE+1:POKE GXSTORE+2
        *NUMSPECTRE,5:POKE GXSTORE+1+2*NUMSPECTRE,1
4050 EVERY 30,1 GOSUB 6000
4999 RETURN
```

Le déroulement du jeu se fait par itération dans le sous-programme 5000 tandis que les deux sous-tâches agissent de manière asynchrone et indépendante. Seules les quatre touches flèches et la touche COPY sont vérifiées ; les autres touches sont ignorées. Le sous-programme s'achève lorsque les deux pierres tombales ont été placées devant les issues, ou lorsque l'indicateur FREE est placé (extérieurement) à l'état faux.

```
5000 REM MOUVEMENT DU HEROS
5010 IF NOT (FREE AND PLAYING) THEN 5999
```

```
5015 Z$=INKEY$:IF Z$=" " THEN 5010
5020 Z=ASC(Z$):IF Z=224 THEN Z=244
5030 IF Z<240 OR Z>244 THEN 5010 ELSE Z=Z-239
5040 NMX=MX:NMY=MY
5050 ON Z GOTO 5100,5200,5300,5400,5500
5100 REM *** HAUT
5110 IF MY>1 THEN NMY=MY-1
5120 GOTO 5600
5200 REM *** BAS
5210 IF MY<25 THEN NMY=MY+1
5220 GOTO 5600
5300 REM *** GAUCHE
5310 IF MX>1 THEN NMX=MX-1
5320 GOTO 5600
5400 REM *** DROITE
5410 IF MX<40 THEN NMX=MX+1
5420 GOTO 5600
5500 REM *** TOUCHE COPY
5510 IF MX=32 AND MY=15 THEN DI:LOCATE MX,MY: PEN
       BOTTLECOLOUR:PRINT EDGE$;:NMX=32:NMY=14:
       GOTO 5530
5520 IF MX=33 AND MY=21 THEN DI:LOCATE MX,MY: PEN
       BOTTLECOLOUR:PRINT EDGE$;:NMX=33:NMY=22:
       GOTO 5530
5525 GOTO 5540
5530 TOMBSTONES=TOMBSTONES+1:IF TOMBSTONES=2 THEN
       PLAYING=FALSE
5540 GOTO 5705
5600 REM *** TEST
5610 DI:LOCATE NMX,NMY:CALL TXTRD : IF
       PEEK(RDCHAR)< >32 THEN EI:GOTO 5720
5700 DI:LOCATE MX,MY:PRINT " ";
5705 LOCATE NMX,NMY:PEN MANCOLOUR:PRINT MAN$;:
       MX=NMX : MY=NMY : EI
5710 POKE MXSTORE,MX: POKE MXSTORE+1,MY
5712 IF MX=2 AND MY=20 THEN DI : STOPPERED=TRUE:
       LOCATE 1,20: PEN BOTTLECOLOUR:PRINT STRING$(
       3,STOPPER$);:NMX=4:NMY=20:GOTO 5705
5720 GOTO 5010
5999 RETURN
```

Le sous-programme 6000 est une sous-tâche indépendante appelée par le sous-programme 4000 (qui fait apparaître les fantômes). Il possède la priorité la plus élevée et les instructions DI/EI sont donc parfaitement superflues.

Sa fonction principale est d'appeler le sous-programme en langage d'assemblage qui déplace les fantômes (voir Chapitre 16).

Il produit également le gémissement des fantômes, en utilisant une enveloppe de ton définie durant la routine d'initialisation.

```
6000 REM MOUVEMENT DES FANTOMES
6010 DI
6020 PEN GHOSTCOLOUR
6030 CALL MVGHOST
6040 IF PEEK(DONEFLAG)=1 THEN FREE=FALSE :
       X=REMAIN(0) : Y=REMAIN(1)
6050 SOUND, 1,350,40,4,0,1
6990 EI
6999 RETURN
```

Bonne chasse aux fantômes !

# IV

# PROGRAMMATION EN LANGAGE D'ASSEMBLAGE

Cette partie donne des informations sur la manière de programmer l'Amstrad en langage d'assemblage ; elle montre comment des sous-programmes peuvent ainsi améliorer les programmes écrits en BASIC.

# 14
# L'ENVIRONNEMENT ZEN

Aussi utile que soit le BASIC, il arrive un moment où nous désirons quelque chose de plus, et nous nous tournons vers l'assembleur. Pour diverses raisons possibles :

- La nécessité d'une plus grande rapidité : les programmes en assembleur peuvent opérer beaucoup plus vite que les programmes en BASIC.
- Le besoin d'économiser de la mémoire : les programmes en assembleur sont beaucoup plus concis que leurs équivalents en BASIC.
- Le besoin d'accéder à des possibilités du système que le BASIC ne peut pas utiliser : toutes les caractéristiques du système sont accessibles au programmeur en assembleur ; il n'est limité que par ses propres connaissances !

Bien entendu, dans sa conception, l'Amstrad est une machine qui ''fonctionne'' en BASIC, puisque le BASIC est disponible en ROM et que la machine charge le langage BASIC dès qu'elle est mise en marche. Mais elle n'est pas limitée au BASIC !

## ZEN

Kuma Computers propose, pour l'Amstrad, un assembleur baptisé simplement ZEN qui permet à l'utilisateur :

- d'écrire des programmes complets en assembleur, qui peuvent être sauvegardés sur cassette et rechargés ultérieurement pour être exécutés indépendamment du BASIC.
- d'écrire des programmes qui seront conçus comme des sous-programmes destinés à être appelés par le BASIC et qui pourront donc ''côtoyer'' les programmes en BASIC.

De plus, ZEN contient un désassembleur, qui peut être utilisé pour éclairer les mystères du système d'exploitation de l'Amstrad si on le désire.

Des versions de ZEN ont été conçues pour d'autres systèmes, comme les ordinateurs Sharp, Newbrain, Epson et MSX ; il s'agit donc d'un produit qui a largement fait ses preuves et qui peut être recommandé à tous les possesseurs d'Amstrad familiarisés avec le langage d'assemblage, ou bien à ceux qui souhaitent l'apprendre. Le manuel fourni contient un listing complet du code source de ZEN ; l'expert peut modifier le produit selon ses besoins et l'étudiant a la possibilité d'examiner un grand programme d'assemblage écrit pour le microprocesseur Z80.

## COMMENT CHARGER ZEN

ZEN est disponible sur cassette. Lorsqu'il est en mémoire, il réside à l'adresse 4000h (adresse de départ), c'est-à-dire en plein milieu de la zone qui est habituellement réservée au BASIC, et c'est pourquoi il est nécessaire d'entrer au préalable une instruction MEMORY pour limiter à 3FFFh (ou 16383) la limite supérieure de la zone du BASIC. ZEN peut alors être chargé par une simple commande LOAD " "et l'on pourra ensuite y accéder en entrant :

```
CALL 16384
```

Il affichera alors son indicatif :

```
ZEN >
```

En vitesse lente, le chargement dure environ une minute.

## TOPOGRAPHIE DE LA MÉMOIRE

A ce moment-là, l'utilisateur souhaite généralement entrer du code source — soit en le tapant, soit en le chargeant à partir d'une cassette — afin de le modifier ou de l'étendre, puis de l'assembler. L'adresse par défaut du code source est 6000h, mais elle peut facilement être changée.

ZEN travaille en mémoire. Si l'on désire exécuter le programme réalisé, et c'est normalement le cas, il faut réserver de la place pour le code objet lorsque ZEN assemble le programme source. Cette zone

du code objet doit en général se trouver au-dessus de celle du code source, disons à l'adresse 8000h.

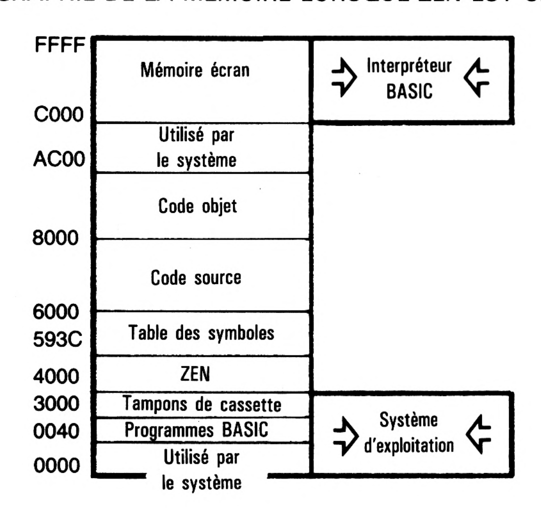
ZEN a besoin d'une zone de travail pour installer la table des symboles contenant tous les noms des variables de la source ; cette table commence juste au-dessus de ZEN lui-même, à l'adresse 593Ch, et s'étend jusqu'au début du code source, situé en 6000h. Si cette zone se révèle insuffisante, l'adresse de départ du code source peut être modifiée.

La seule autre zone de la mémoire qui doit être mentionnée est celle constituée par les 4K nécessaires à la lecture et à l'écriture des cassettes. Quand vous entrez l'instruction de chargement (LOAD) de ZEN, le BASIC fait son tour de passe-passe habituel et réduit HIMEM de 4096 octets, lui donnant dans ce cas la valeur 2FFFh. ZEN s'attend donc à trouver un tampon *(buffer)* de lecture en 3000h et un tampon d'écriture en 3800h. Si vous voulez les placer ailleurs dans la mémoire, ZEN en sera très satisfait, à condition bien sûr que vous l'informiez de la modification des deux pointeurs.

La position du pointeur est contrôlée par l'utilisateur ; il est donc parfaitement possible d'avoir plus d'un programme source en mémoire — chaque programme source étant terminé par une instruction END — et d'assembler alternativement l'un ou l'autre.

En résumé, voici la topographie normale de la mémoire lorsque ZEN est chargé. Comparez cette figure avec celle qui est présentée dans le Chapitre 1.

TOPOGRAPHIE DE LA MÉMOIRE LORSQUE ZEN EST CHARGÉ



Il faut bien préciser que cette topographie est celle d'une utilisation normale, mais qu'elle n'est pas absolue. Des programmes BASIC plus importants peuvent y trouver leur place si l'on repousse les tampons de cassette au-dessus de la zone occupée par ZEN, et les limites des zones réservées au code source et au code objet peuvent être modifiées par l'utilisateur.

En général, la taille de la mémoire dédiée au BASIC ne constitue pas un problème. Il existe sur le marché bien d'autres systèmes qui seraient heureux d'en avoir autant ! Si vous voyez que votre programme BASIC est trop grand pour coexister en mémoire avec ZEN, n'oubliez pas que, lorsque vous avez testé votre routine en assembleur grâce à un minuscule programme BASIC, le programme objet peut être chargé sans que la présence de ZEN ou du code source soit nécessaire, et il pourra être appelé directement par votre programme BASIC — dans ce cas, la valeur de HIMEM sera probablement égale ou supérieure à 7FFFh.

## LES COMMANDES DE ZEN

Après avoir chargé ZEN en mémoire, comme indiqué précédemment, de quelles commandes disposez-vous ? Elles sont toutes représentées par une seule lettre, certaines étant suivies d'un paramètre.

| | | | |
|---|---|---|---|
| A | Assemble | O | Out |
| B | Bye (Retour au BASIC) | P | Print |
| C | Copy | Q | Query |
| D | Down | R | Read |
| E | Enter | S | Sort |
| F | Fill | T | Target |
| G | Goto | U | Up |
| H | Howbig | W | Write |
| I | In | X | eXamine |
| K | Kill | Z | Zap |
| L | Locate | c | Catalogue |
| M | Modify | d | Disassemble |
| N | New | u | Unscramble |

Vous pouvez constater que ZEN nous offre les bases d'un contexte de développement complet. Toutes les opérations d'assemblage sont

faites directement en mémoire, et le code objet est aussitôt disponible, en cas de besoin.

Les programmes source sont entrés grâce à la commande E, assemblés par la commande A, et peuvent être effacés par la commande Kill (K). Quand l'assemblage est "propre", ils peuvent être exécutés par une commande Goto (G). Un point d'arrêt peut être déterminé pour que l'exécution se fasse par étapes, en affichant l'état des registres et de la mémoire à chaque halte.

ZEN ne nous offre pas un éditeur d'écran complet pour entrer le code source, mais des commandes nous permettent d'afficher une ligne ou plusieurs lignes (P), de localiser une chaîne de caractères spécifique (L), de nous rendre à une ligne particulière (T), de faire monter ou descendre sur l'écran un certain nombre de lignes (U et D), ou d'effacer une ou plusieurs lignes (Z).

Les données peuvent être créées par Fill (F), Copy (C) ou Modify (M) ; elles peuvent être examinées par Query (Q) ou X (qui affiche l'état des registres).

Vous pouvez sauvegarder des fichiers source ou des fichiers objet grâce à la commande W, et les relire avec la commande R. La liste des fichiers de la cassette peut être donnée par la commande catalogue (c — "c" minuscule).

Le désassembleur est appelé par la commande d, et vous disposez également d'une option très utile de mise au point qui peut désassembler 8 instructions à la fois.

Les résultats de la commande Assemble (A), de la commande de tri et d'affichage de la table des symboles (S), et du désassembleur (d) peuvent être affichés sur l'écran ou envoyés vers un appareil extérieur, c'est-à-dire une imprimante.

L'utilisateur peut se servir de la commande B pour passer de ZEN en BASIC, il pourra ensuite entrer CALL 16384 pour revenir à ZEN. Dans une zone comme dans l'autre, vous retrouverez les programmes tels que vous les avez laissés.

## RÉSUMÉ

ZEN est un produit fort utile, qui offre un environnement compréhensible et bien documenté à ceux qui souhaitent programmer en assembleur. La connaissance du langage d'assemblage constitue un capital énorme et nous ouvre bien des portes, c'est pourquoi nous invitons les hésitants à plonger. L'eau est bonne !

# 15

# INTERFACE AVEC LE BASIC

La plupart des utilisateurs qui approchent l'assembleur pour la première fois ont l'intention d'écrire des sous-programmes élaborés à partir du BASIC plutôt que des programmes d'assemblage indépendants. Dans bien des cas, des paramètres devront être transférés d'un langage à l'autre. Ce chapitre examine les techniques permettant de passer tranquillement du BASIC à l'assembleur, puis d'y revenir.

Les exemples donnés s'appliquent particulièrement à l'environnement de l'assembleur ZEN, mais si un autre assembleur est utilisé, les différences devraient être minimes, car ce que nous étudions en réalité, ce sont les conventions du BASIC de l'Amstrad.

## COMMENT DÉVELOPPER UN SOUS-PROGRAMME

Les programmeurs en BASIC ont l'habitude de trouver et de corriger des fautes dans leurs programmes ; bien peu d'entre nous peuvent se targuer d'écrire un programme important sans faire de fautes. Lorsque nous exécutons le programme et qu'une erreur est révélée — une boucle sans fin, par exemple — nous arrêtons l'exécution, corrigeons l'erreur, et relançons le programme. S'il s'agit d'une division par zéro, nous effectuons une vérification supplémentaire pour nous assurer que cette situation ne peut pas se reproduire.

Avec l'assembleur, une erreur de programmation peut effacer en quelques microsecondes le travail de toute une soirée !

C'est la raison pour laquelle nous invitons les utilisateurs à effectuer fréquemment des sauvegardes de leurs programmes source. Le code objet peut être restauré en quelques secondes si la source est intacte. Par contre, le remplacement du code source peut prendre des heures.

Voici une suite d'opérations que nous vous proposons d'effectuer lorsque vous développez des sous-programmes en assembleur avec ZEN. Cela vous évitera de vieillir avant l'âge...

1. Tapez une commande MEMORY 16383 et chargez l'assembleur ZEN au-dessus du BASIC.
2. Entrez ZEN et tapez le code source.
3. Créez un programme subsidiaire — qui pourra éventuellement être abandonné — à côté du programme principal, afin d'y installer les registres et les données qui apparaîtront lorsque le sous-programme sera appelé à partir du BASIC.
4. Procédez à l'assemblage et débarrassez-vous des erreurs telles que fautes d'orthographe, symboles non déclarés, etc.
5. Sauvegardez (deux fois) le code source sur la cassette. METTEZ UNE ÉTIQUETTE SUR LA CASSETTE !
6. Lancez le programme subsidiaire et arrêtez-le en plaçant un point d'arrêt au début du sous-programme principal ; vérifiez qu'il installe l'environnement comme prévu.
7. Relancez le programme subsidiaire, en plaçant cette fois le point d'arrêt plus loin dans le programme. Vérifiez que les zones des registres et des données sont correctement manipulées.
8. Si (ou plutôt lorsque !) une erreur se produit, notez les corrections à faire par n'importe quel moyen — en faisant une note au crayon sur le dos d'une enveloppe, ou en envoyant un nouvel assemblage directement vers l'imprimante, en indiquant la date et l'heure.
9. Modifiez le numéro de la version (qui devrait se trouver dans une ligne de commentaire au début du programme) et faites une nouvelle sauvegarde dès que vous avez changé quelques instructions. NE SAUVEGARDEZ PAS CETTE VERSION PAR DESSUS LA DERNIÈRE COPIE DU PROGRAMME SOURCE ; utilisez une autre cassette, pour avoir au moins deux cassettes de travail. Indiquez l'heure dans le titre du programme que vous sauvegardez, comme ceci :

   SAVE ''ABC3 4.30 23/8'',B,...

10. Quand le programme semble fonctionner tout à fait correctement, retournez au BASIC avec la commande BYE, écrivez un programme BASIC pour appeler le sous-programme et recommencez les tests en utilisant toutes les combinaisons de données.

11. Revenez à ZEN et effacez le programme subsidiaire qui installe les données. Puis effectuez un réassemblage.

12. Sauvegardez le code objet (pour la première fois) sur cassette.

13. Réinitialisez l'ensemble du système. (Vous pourrez le faire en sautant à l'adresse 0, par exemple !)

14. Entrez le programme BASIC pour définir les limites de la mémoire et chargez le module objet en assembleur. Appelez ensuite le module pour vérifier que l'opération est correcte.

Si vous trouvez cela plus difficile que la programmation en BASIC, vous avez parfaitement raison ! Mais il y a néanmoins des avantages, comme vous vous en rendrez compte lorsque votre premier programme finira par fonctionner comme vous le souhaitez...

## UN MOYEN SIMPLE DE FAIRE PASSER DES INFORMATIONS

Prenons comme exemple un sous-programme simple. Le système d'exploitation de l'Amstrad est plein de routines très utiles que nous pouvons appeler directement à partir du BASIC, ou utiliser avec des sous-programmes en assembleur. Une liste des routines les plus intéressantes est donnée dans le prochain chapitre, et une liste complète se trouve dans le manuel technique de l'Amstrad.

Si nous voulons, par exemple, lancer une balle sur l'écran, nous souhaitons bien sûr savoir si nous allons rebondir sur quelque chose.

Cela semble constituer un problème ! En BASIC, aucune commande ne nous indique quel caractère se trouve à un emplacement donné de l'écran ; nous pourrions utiliser TEST ou TESTR pour vérifier la présence d'une couleur spécifique, mais cela serait insuffisant dans de nombreuses applications. Nous pourrions lire la mémoire écran (avec PEEK), mais sa topographie est très complexe et le déchiffrage serait extrêmement difficile. Cependant, il existe une routine du système d'exploitation appelée TXT RD CHAR qui correspond exactement à ce que nous souhaitons.

Pour utiliser cette routine (ce sous-programme), nous devons nous placer sur le "carré" en question, grâce à la commande LOCATE, puis appeler la routine qui donnera au registre A la valeur du caractère — 32 pour un espace, 65 pour un "A" majuscule, 250 pour une silhouette humaine, etc. Si aucun caractère valide ne se trouve à cet

emplacement (ce qui peut arriver lorsqu'un caractère se superpose à un autre), l'indicateur de retenue *(carry flag)* est mis à zéro et le registre A donne zéro.

Il nous faut donc un petit sous-programme en assembleur que nous pourrons appeler à partir du BASIC, et qui appellera à son tour le sous-programme TXT RD CHAR afin de placer le contenu du registre A à une adresse de la mémoire où le BASIC pourra le lire.

Cet échange d'information peut se faire de deux manières :

- en passant le nom d'une variable du BASIC contenant le caractère ;
- en écrivant le sous-programme en assembleur afin qu'il puisse placer le caractère à une adresse précise que le BASIC pourra lire avec une instruction PEEK.

Cette seconde méthode n'est pas très élégante, mais elle est remarquablement facile ; c'est celle que nous présenterons tout d'abord.

Normalement, HIMEM a pour valeur AB7Fh (ou 43903, si vous préférez). Notre sous-programme sera probablement court ; nous l'installerons en AB60h et nous lui ferons placer le caractère en AB7Fh.

La routine TXT RD CHAR est située en BB60h. Elle n'exige pas de paramètres d'entrée, mais deux résultats sont possibles :

- Retenue mise à 1 et caractère dans le registre A.
- Retenue mise à 0 et registre A mis à 0 (erreur).

Pour les besoins de notre exemple, nous ignorerons le cas d'erreur et supposerons que le caractère retourné par notre routine est toujours correct. Voici cette routine, entrée avec ZEN.

```
ORG   0AB60H          ;Position de départ
LOAD  $               ;Génération du code objet
CALL  0BB60H          ;Appeler TXT RD CHAR
LD    (0AB7FH),A      ;Stocker le résultat
RET                   ;Retourner au BASIC
END
```

L'assemblage de cette routine n'occupe que 7 petits octets (nous avions bien dit que l'assembleur était concis !) à partir de l'adresse 0AB60h.

Si nous suivions notre propre mise en garde, telle qu'elle a été donnée précédemment, nous devrions effectuer toute une série d'opérations pour tester notre routine ; cependant, les règles sont faites pour ne pas être suivies aveuglément, et nous testerons ce sous-programme grâce au BASIC, car il est plus facile de se déplacer sur l'écran avec l'instruction LOCATE.

Retournons au BASIC et entrons le programme suivant :

```
10 CALL &AB60
20 PRINT PEEK (&AB7F)
```

Effaçons l'écran et lançons le programme. Vous devriez voir affichée la valeur 32, qui est celle d'un espace. Comme RUN, suivi de ENTER, a été tapé à la deuxième ligne de l'écran, le système a placé le curseur sur la première colonne de la troisième ligne de l'écran, où est affiché un espace.

Parfait ! Maintenant, sans effacer l'écran, entrons :

```
5 LOCATE 2,3
```

et tapons de nouveau RUN. Cette fois, vous devriez voir affichée la valeur 51, car le caractère qui se trouve sur la deuxième colonne de la troisième ligne (où nous avons placé le curseur) est un "3", et, comme vous pouvez le vérifier en examinant la table des caractères du *Guide de l'utilisateur* (Appendice III — p. A3.1), le chiffre 3 a la valeur 51 dans le code ASCII.

Vous pouvez imaginer d'autres tests ; ils devraient nous prouver que, tant que nous nous plaçons à une position correcte, le sous-programme fonctionne comme nous le souhaitons. Bravo !

## UTILISONS DES PARAMÈTRES

Maintenant que nous disposons de notre première routine en assembleur, nous allons devenir ambitieux et placer le résultat dans une variable plutôt que dans une adresse fixe de la mémoire. Ce que nous voulons, c'est vérifier qu'un espace se trouve à une position donnée en tapant :

```
CALL &AB60,A$
IF A$< >"  " THEN...
```

Mais cela ne marchera pas, quelles que soient les modifications que nous pourrions apporter à notre sous-programme en assembleur ! La raison en est que les paramètres du BASIC de l'Amstrad sont toujours passés *en tant que valeurs.* Cela signifie que, dans l'exemple précédent, le BASIC va stocker quelque part une copie de A$ et la passer au sous-programme. Comme il s'agit d'une copie, vous pouvez la modifier au point de devenir rouge de fureur, cela n'altérera en rien le contenu original de A$, qui demeure inchangé.

Heureusement, il existe une solution simple. Le BASIC de l'Amstrad nous offre un opérateur peu connu appelé '' @ '' qui signifie ''au lieu de passer une copie de la valeur de la variable, passer l'adresse de la variable''. Nous pouvons ainsi utiliser l'adresse pour accéder à la variable originale et la modifier. Avant d'appeler la routine, notre programme en BASIC doit définir une constante ayant une longueur de 1 (car l'assembleur ne peut pas altérer la longueur d'une variable du BASIC). Le programme de vérification sera :

```
10 A$="Z"
20 CALL &AB60, @A$
30 PRINT A$
```

et nous verrons la variable A$ passer de la valeur ''Z'' à une autre valeur.

## TYPES DE PARAMÈTRES

Avant d'indiquer les changements du sous-programme en assembleur, nous devons faire une différence entre les divers types de paramètres, et montrer comment ils sont stockés en BASIC.

Il y a trois types de variables : les chaînes, les nombres entiers et les nombres réels.

Comme vous le savez, les chaînes de caractères sont généralement représentées par un nom qui se termine par le caractère ''$'' (bien que ce ne soit pas toujours le cas — voir la commande DEFSTR). Ils peuvent avoir jusqu'à 255 octets de long. Une variable chaîne est en fait stockée en deux parties, le *descripteur de chaîne* et le *corps de chaîne.* Le descripteur contient la longueur (en un seul octet, et c'est pourquoi sa valeur maximale est 255) suivie de l'adresse du corps. Le corps contient les données de la variable.

La valeur d'un entier peut aller de − 32768 à + 32767. Si vous connaissez le langage d'assemblage du Z80, vous comprendrez ce que cela signifie : cette valeur est stockée dans 2 octets.

Les réels sont utilisés pour des valeurs plus grandes que celles qui peuvent être attribuées aux entiers, et pour indiquer les fractions. Leur représentation interne est assez compliquée car elle utilise une mantisse et un exposant.

## PARAMÈTRES : LES CHAINES

Sachant cela, nous voudrions que la chaîne A$ de notre exemple soit stockée (après la ligne 10) de cette manière :

```
Descripteur   —  DB   1       ;longueur
              —  DW   corps   ;adresse du corps

Corps         —  DB   'Z'
```

Que le paramètre soit une valeur ou une adresse, le registre IX "pointe" vers lui dès l'entrée du sous-programme. Notre sous-programme en assembleur pourrait donc être modifié comme ceci :

```
ORG   0AB60H
LOAD  $
CALL  0BB60H     ;APPELER TXT RD CHAR
LD    L,(IX+0)   ;ADRESSE DU DESCRIPTEUR DE A$
LD    H(IX+1)
INC   HL         ;IGNORER LA LONGUEUR
LD    E,(HL)     ;LIRE L'ADRESSE DU CORPS
INC   HL
LD    D,(HL)     ;DANS DE
LD    (DE),A     ;STOCKER VALEUR DE A DANS LE
                  CORPS
RET
```

et le sous-programme en BASIC devrait être modifié comme précédemment pour inclure l'opérateur " @ ".

Vous devriez pouvoir vous placer en divers endroits de l'écran pour vérifier que le sous-programme fonctionne correctement.

## PARAMÈTRES : LES ENTIERS

Le format du paramètre d'un entier est plus simple. La valeur est contenue sous la forme d'un nombre binaire signé de 16 bits, ayant une valeur comprise entre −32768 et +32767 (utilisant une notation en complément à deux dans le cas des valeurs négatives). Une fois encore, si nous voulons modifier la valeur de la variable passée durant le sous-programme, nous devons passer son adresse en utilisant l'opérateur " @ ". Si nous souhaitons simplement lire la valeur sans la modifier, il est inutile de nous encombrer du " @ ". Notre sous-programme pourrait donc utiliser un nombre entier plutôt qu'une variable chaîne :

```
10 Z%=0
20 CALL &AB60, @ Z%
30 PRINT Z%
```

et le sous-programme en assembleur pourrait être modifié ainsi :

```
ORG     0AB60H
LOAD    $
CALL    0BB60H
LD      L,(IX+0)
LD      H,(IX+1)
LD      (HL),A
RET
```

Effectuez les vérifications habituelles.

## PARAMÈTRES : LES RÉELS

Ayant maîtrisé les chaînes et les entiers, il nous reste les réels. Malheureusement, leur format est complexe ! Un exposant de 4 octets est suivi par une mantisse de 1 octet qui est "décalée de 128". Ce format est expliqué plus en détail dans l'*Amstrad Concise BASIC Spe-*

*cific Specification* (SOFT 157) qui décrit le BASIC de l'Amstrad d'une manière plus approfondie que le guide standard. Si vous désirez étudier le format de divers nombres réels, nous vous suggérons d'écrire un sous-programme qui place les 5 octets dans des zones spécifiques de la mémoire, puis affiche leur valeur en revenant au BASIC. Ce programme suffira :

```
10 Z=32
20 CALL &AB60, @ Z
30 FOR J=&AB7B TO &AB7F
40 PRINT HEX$(PEEK(J));" ";
50 NEXT J
60 PRINT
```

et vous l'associerez à ce sous-programme en assembleur :

```
ORG     0AB60H
LOAD    $
CALL    0BB60H
LD      L,(IX+0)
LD      H,(IX+1)
LD      DE,0AB7BH
LD      BC,5
LDIR
RET
```

En modifiant la ligne 10, vous obtiendrez des résultats de ce genre :

```
0  0  0  0   0     z = 0
0  0  0  0   81    z = 1
0  0  0  0   82    z = 2
0  0  0  80  81    z = -1, etc.
```

Dans cet exemple, nous avons passé l'adresse du nombre réel et non sa valeur elle-même, grâce à l'opérateur @ . Il faut préciser que si vous tentez de passer la valeur elle-même, le BASIC convertit le nombre réel en un nombre entier non signé, et que vous passez alors un nombre de 2 octets et non pas un nombre complexe occupant

5 octets comme celui que nous venons de montrer. Bien entendu, la conversion d'un nombre réel en nombre entier entraînera généralement une perte d'information car les fractions seront arrondies à l'entier le plus proche.

A moins d'écrire vos propres routines mathématiques en assembleur pour calculer ces nombres complexes, vos paramètres seront sans doute rarement des nombres réels !

## PLUSIEURS PARAMÈTRES

Pour l'instant, nous avons uniquement étudié le moyen de faire passer un paramètre. Mais il est possible d'en faire passer plusieurs, et nous pouvons mêler des *valeurs* et des *adresses* si nous le désirons. Quel que soit le type choisi, le paramètre sera toujours représenté par un nombre de 2 octets.

Lorsque vous entrez le sous-programme, le BASIC place dans le registre A le nombre de paramètres (à partir de 0). Comme nous l'avons déjà vu, le registre IX est défini de manière à pointer vers les paramètres eux-mêmes, mais notez bien qu'il pointe vers le dernier, et non pas vers le premier. Pour illustrer cela, examinons le fragment suivant :

```
 5 A%=5 : ABC$="HELLO" : Z=3.5
10 CALL &8000, @ A%, @ ABC$,Z, -1
```

Il y a quatre paramètes : deux adresses et deux valeurs. Ainsi, la valeur initiale de A sera 4 et le registre IX sera chargé comme ceci :

```
Registre IX ——► DW  -1
                DW  valeur de Z
                DW  adresse du descripteur de ABC$
                DW  adresse de A%
```

ce qui est peut-être l'inverse de ce que vous attendiez.

Quand nous avons testé le programme précédent, nous avons enregistré les valeurs suivantes ; si vous entrez les mêmes valeurs, il y aura peut-être des petites différences (selon votre version du BASIC et de ZEN, ou si vous tapez des espaces supplémentaires), mais les résultats devraient être identiques.

Le registre IX contenait BFF6h. Et à l'adresse BFF6h, nous trouvons :

```
BFF6:  FF FF
BFF8:  04 00 CA 01 C2 01 ...
```

et vous pouvez constater que les paramètres sont tous stockés en tant que nombres de 2 octets, et que IX pointe bien vers – 1 (FFFFh), suivi de 4 (Z converti à l'entier le plus proche), suivi des deux adresses 01CAh et 01C2h. En 01CAh nous trouvons le descripteur de chaîne contenant la longueur, 05h, suivie par une adresse, 0185h, qui pointe correctement vers le corps de la chaîne ABC$ :

```
0185:  48 45 4C 4C 4F     HELLO
```

Finalement, à l'adresse 01C2h, nous trouvons 05 00 qui est la valeur de A%.

Dans cet exemple, nous pourrions modifier les valeurs de A% et de ABC$, mais non pas celle de Z ni (bien évidemment) celle de – 1.

Tout est bien clair ?

# 16

# LES ROUTINES DU SYSTÈME D'EXPLOITATION

Maintenant que vous pouvez établir des relations entre le BASIC et les routines en assembleur, vous êtes certainement désireux de tester quelques idées.

## BLOCS JUMP

Le système d'exploitation de l'Amstrad a été conçu pour être accessible aux utilisateurs. Il contient un grand nombre de sous-programmes qui lui sont directement nécessaires pour le bon fonctionnement de l'ordinateur, mais qui sont également documentés et accessibles aux programmes de l'utilisateur.

Un problème qui se pose toujours aux concepteurs est introduit par le fait que les sous-programmes peuvent se trouver à des emplacements différents selon les versions du système d'exploitation, en fonction des corrections et des améliorations qui lui sont apportées. Si l'utilisateur écrit son programme en pensant que la routine TXT RD CHAR se trouve à l'adresse BB60h et découvre qu'elle se trouve en BB68h dans une version ultérieure de l'Amstrad, il risque d'être plutôt mécontent, tout comme les autres utilisateurs de son programme.

Dans le cas de l'Amstrad, ce problème est résolu par des "blocs JUMP". Comme leur nom l'indique, les blocs JUMP sont tout bonnement des suites d'instructions JUMP qui envoient vers diverses routines du système d'exploitation. Ces blocs sont contenus dans les ROM et copiés en RAM lors de l'initialisation du système. Ils sont toujours placés au même endroit, et l'utilisateur peut ainsi être sûr qu'il y aura toujours une instruction TXT.RD.CHAR à l'adresse BB60h, tandis que le concepteur est libre de déplacer TXT RD CHAR là où il le désire, pourvu qu'il modifie le bloc JUMP en conséquence. C'est simple !

La morale de cette histoire : fiez-vous aux blocs JUMP, et ne cherchez pas à les éviter. Pensez qu'au bout de deux ans, un Amstrad

disposant d'un lecteur de disquette et de ROM supplémentaires pourra posséder une mémoire très différente, mais que les blocs JUMP seront certainement identiques.

Il y a quatre blocs JUMP au total. Le manuel technique de l'Amstrad les détaille d'une manière approfondie et insiste sur les conditions d'entrée et de sortie de chaque routine. Si vous envisagez de programmer sérieusement en assembleur, la lecture du manuel technique vous sera absolument nécessaire. Nous ne cherchons pas à le remplacer ici, mais seulement à préciser quelles sont les routines les plus importantes.

Les utilisateurs confirmés peuvent noter qu'il leur est possible de modifier une instruction de saut (JUMP) d'un bloc JUMP afin de pointer vers leur propre routine en RAM plutôt que vers la routine initiale du système, qui se trouve en ROM. Si les conditions d'entrée et de sortie sont respectées par la nouvelle routine, d'autres programmes peuvent l'utiliser de manière satisfaisante. Mais vous devez être sûr de ce que vous faites...

## LE BLOC JUMP PRINCIPAL

Le bloc JUMP principal — celui auquel les utilisateurs souhaitent généralement avoir accès — se situe en BB00h - BD37h. Il contient 190 instructions de 3 octets, mais en fait la plupart d'entre elles ne sont pas des sauts ! Vous vous souvenez que les ROM de l'Amstrad se superposent à la RAM aux deux extrémités de la mémoire. La plupart des rubriques du bloc JUMP principal sont en fait composées d'une instruction RST 8 suivie par l'adresse d'une routine, et qui place les indicateurs dans les bits les plus significatifs. Cela provoque le saut demandé vers le bas de la mémoire, mais permet également de valider ou d'inhiber les ROM, selon l'état des indicateurs. Il s'agit en effet d'une instruction supplémentaire du Z80 qui nous dit : "effectuer un saut et déterminer l'état de la ROM".

Certaines des rubriques les plus utiles de ce bloc JUMP sont les suivantes :

- **BB00 Keyboard Manager Initialise** est utilisée pour remettre le clavier dans son état initial (qui suit la mise en route du système). Toutes les expansions retrouvent leurs valeurs "par défaut", le tampon (buffer) est vidé, les vitesses de répétition retrouvent leur valeur initiale, etc.

- **BB06 Keyboard Wait Char** attend le prochain caractère venant du clavier. C'est généralement le prochain caractère tapé, mais ce peut être, par exemple, le prochain caractère d'une chaîne d'expansion, ou un caractère d'une "table de traduction" de touches. Le registre A contient le caractère.

- **BB09 Keyboard Read Char** Contrairement à la routine précédente, celle-ci retourne immédiatement une valeur, qu'un caractère soit prêt ou non. Si un caractère est trouvé, il est placé dans A et l'indicateur de retenue est mis à 1. Sinon, la retenue est mise à 0.

- **BB24 Get Joystick** détermine l'état des joysticks. Un octet d'information est donné pour chaque joystick (qu'il soit ou non branché sur l'Amstrad). Le registre H contient la valeur du joystick 0 et le registre L la valeur du joystick 1. Les bits sont :

  0 — haut
  1 — bas
  2 — gauche
  3 — droite
  4 — FIRE 2
  5 — FIRE 1

- **BB4E Txt Initialise** Initialisation complète du texte de l'unité de visualisation, comme lors d'une remise en marche du système. Tous les "flux" (canaux) retrouvent leur valeur par défaut, l'écran est effacé et le curseur est placé dans le coin supérieur gauche.

- **BB5A Txt Output** envoie un caractère vers le canal actuellement sélectionné. Le caractère, contenu dans le registre A, peut être un caractère de contrôle, auquel cas il n'est pas affiché.

- **BB5D Txt Wr Char** écrit un caractère par le canal actuellement sélectionné. Contrairement à la routine Txt Output, les caractères de contrôle sont affichés comme les autres.

- **BB60 Txt Rd Char** lit un caractère à partir de l'écran. Si vous avez utilisé les exemples de ce livre, vous devriez bien connaître cette routine ! Si un caractère valide se trouve à la position sélectionnée, il est placé dans le registre A, avec l'indicateur de retenue à 1. Sinon, l'indicateur de retenue est mis à 0.

- **BB63 Txt Set Graphic** Cette routine est à la fois utilisée pour valider

et inhiber l'écriture des caractères graphiques. La valeur de A est différente de zéro pour la validation, et zéro pour l'inhibition. Cela équivaut à l'utilisation de TAG et de TAGOFF en BASIC.

- **BB6C Txt Clear Window** efface la fenêtre sélectionnée. Elle retrouve la valeur de l'encre du papier du flux actuellement sélectionné.
- **BB6F Txt Set Column** Le registre A contient le numéro de la colonne sélectionnée. Cette routine donne la position horizontale du curseur.
- **BB72 Txt Set Row** Le registre A contient le numéro de la ligne.
- **BB75 Txt Set Cursor** Le registre H contient le numéro de la colonne choisie et le registre L le numéro de la ligne choisie. Cette routine accomplit une tâche équivalente de celles des deux routines précédentes.
- **BB90 Txt Set Pen** Le registre A contient la valeur de l'encre à utiliser. Le système masque cette valeur pour s'assurer qu'elle ne dépassera pas les limites autorisées.
- **BB96 Txt Set Paper** Le registre A contient la valeur de l'encre attribuée au papier.
- **BB9F Txt Set Background** Si A est égal à zéro, la transparence est inhibée. Si A est différent de zéro, la transparence est validée. Cela affecte l'écriture des caractères sur l'écran en mode texte ; si la transparence est validée, le fond n'est pas préalablement effacé et les caractères sont écrits sur ce qui se trouve déjà sur l'écran. Cette routine est surtout utilisée pour sous-titrer des diagrammes, etc. (*Remarque :* En mode graphique, le fond est TOUJOURS effacé préalablement, que la transparence soit validée ou non.)
- **BBB4 Txt Stream select** Le registre A contient le numéro du flux (canal), entre 0 et 7, qui est sélectionné pour l'écriture ultérieure.
- **BBBA Gra Initialise** Le flux graphique de l'unité de visualisation est initialisé, comme lors d'une mise en marche du système.
- **BBC0 Gra Move Absolute** Le registre DE contient l'adresse X sélectionnée (position horizontale) et le registre HL contient l'adresse Y. Ces positions peuvent se trouver en dehors de l'écran.
- **BBDB Gra Clear Window** La fenêtre graphique est effacée et prend la couleur de l'encre attribuée au papier graphique.

- **BBDE Gra Set Pen** L'encre graphique prend la valeur de l'encre contenue dans le registre A.
- **BBE4 Gra Set Paper** La couleur du fond est initialisée par la valeur contenue dans A.
- **BBEA Gra Plot Absolute** Un point est affiché à la position déterminée par la coordonnée X (contenue dans DE) et par la coordonnée Y (contenue dans HL).
- **BBF6 Gra Line Absolute** Une ligne est dessinée jusqu'à la position déterminée par les coordonnées X et Y (contenues dans DE et HL).
- **BBFC Gra Wr Char** Un caractère, contenu dans le registre A, est écrit sur l'écran à la position du curseur graphique.
- **BC14 Scr Clear** L'écran est effacé et prend la couleur de l'encre 0.

Évidemment, il y en a bien d'autres, et nous pourrions donner de nombreux détails supplémentaires sur l'utilisation de ces routines. Consultez le manuel technique pour des explications plus approfondies. Les indications précédentes ont pour but de vous faire entrevoir tout ce à quoi vous pouvez avoir accès.

## UN EXEMPLE DE SOUS-PROGRAMME

Le sous-programme suivant est tiré du programme ''Fantômes en bouteille'' (Chapitre 13). Il montre comment sont appelées trois des routines décrites précédemment, et vous propose une manière possible de relier vos programmes en BASIC et des routines du système d'exploitation.

Ce sous-programme contient en fait deux routines. La première, à l'adresse 7000h, lit le caractère situé à la position actuelle du curseur, stocke le caractère dans SAVECHAR (7007h) et retourne au BASIC. Le BASIC peut récupérer le caractère en lisant l'adresse 7007h (grâce à une commande PEEK).

La seconde routine, beaucoup plus longue, s'occupe du déplacement des fantômes sur l'écran. Sa tâche consiste à lire successivement la position de chaque fantôme et à le déplacer en direction du héros (dont la position est déterminée par les coordonnées MX et MY que le BASIC a placées en RAM grâce à des commandes POKE).

Les coordonnées de chaque fantôme sont modifiées selon que le héros se trouve plus haut ou plus bas, plus à droite ou plus à gauche. Cela nous donne une nouvelle série de coordonnées qui sont placées sur la pile par l'instruction G4. Le curseur est installé dans la nouvelle position et le caractère qui s'y trouve déjà est examiné. S'il s'agit du héros, la poursuite est terminée et l'indicateur DONEFLAG est mis à l'état 1 pour informer le BASIC de la situation. Si ce n'est pas un espace, aucune action n'est accomplie. Mais si la voie est libre, un espace est affiché à la position actuelle du fantôme et ce dernier est redessiné à sa nouvelle position. Les nouvelles coordonnées sont alors stockées pour la prochaine exécution de la routine.

```
                MAN:        EQU  249
                GHOST:      EQU  225
                SPACE:      EQU  32
                            ORG  7000H
                            LOAD $
                TXTSETCUR:  EQU  0BB75H
                TXTWRCHAR:  EQU  0BB5DH
                TXTRDCHAR:  EQU  0BB60H
7000 CD60BB     RD:         CALL TXTRDCHAR
7003 320770                 LD   (SAVECHAR),A
7006 C9                     RET
7007 00         SAVECHAR:   DB   0
                MVGHOSTS:   EQU  $
7008 AF                     XOR  A
7009 32FE70                 LD   (DONEFLAG),A       ;attente
700C 3AFF70                 LD   A,(NUMGHOSTS)
700F 32FD70                 LD   (TEMPGHOSTS),A
7012 2A0071                 LD   HL,(MX)            ;placer coord. Man dans HL
7015 E5                     PUSH HL
7016 C1                     POP  BC                 ;changer en BC
7017 210271                 LD   HL,G1X
701A C5         G0:         PUSH BC
701B 56                     LD   D,(HL)             ;D= GX
701C 23                     INC  HL
701D 5E                     LD   E,(HL)             ;E= GY
701E 23                     INC  HL
701F E5                     PUSH HL                 ;Ptr vers nouvelles coord.
7020 D5                     PUSH DE                 ;nouvelles coord. fantôme
7021 79                     LD   A,C
7022 BA                     CP   D
7023 2806                   JR   Z,G2
7025 3803                   JR   C,G1
7027 14                     INC  D                  ;GX+1
7028 1801                   JR   G2
702A 15         G1:         DEC  D                  ;GX-1
702B 78         G2:         LD   A,B                ;MY
702C BB                     CP   E
```

```
7020 2806                  JR   Z,G4
702F 3803                  JR   C,G3
7031 1C                    INC  E               ;GY+1
7032 1801                  JR   G4
7034 1D          G3:       DEC  E               ;GY-1
7035 EB          G4:       EX   DE,HL
7036 E5                    PUSH HL              ;sauvegarder NGX,NGY
7037 CD75BB                CALL TXTSETCUR
703A CD60BB                CALL TXTRDCHAR
703D D1                    POP  DE
703E E1                    POP  HL
703F D5                    PUSH DE
7040 FEF9                  CP   MAN
7042 200A                  JR   NZ,NOTDONE
                 DONE:     EQU  $
7044 3E01                  LD   A,1
7046 32FE70                LD   (DONEFLAG),A
7049 32FD70                LD   (TEMPGHOSTS),A
704C 1804                  JR   WRITE
                 NOTDONE:  EQU  $
704E FE20                  CP   SPACE
7050 201C                  JR   NZ,NOWRITE
                 WRITE:    EQU  $
7052 CD75BB                CALL TXTSETCUR
7055 3E20                  LD   A,SPACE
7057 CD5DBB                CALL TXTWRCHAR
705A E1                    POP  HL
705B E5                    PUSH HL
705C CD75BB                CALL TXTSETCUR
705F 3EE1                  LD   A,GHOST
7061 CD5DBB                CALL TXTWRCHAR
7064 D1                    POP  DE
7065 E1                    POP  HL
7066 2B                    DEC  HL
7067 2B                    DEC  HL
7068 72                    LD   (HL),D
7069 23                    INC  HL
706A 73                    LD   (HL),E
706B 23                    INC  HL
706C E5                    PUSH HL              ;Ptr vers nouvelles coord.
706D D5                    PUSH DE
                 NOWRITE:  EQU  $
706E 3AFD70                LD   A,(TEMPGHOSTS)
7071 3D                    DEC  A
7072 32FD70                LD   (TEMPGHOSTS),A
7075 D1                    POP  DE
7076 E1                    POP  HL
7077 C1                    POP  BC
7078 20A0                  JR   NZ,GO
707A C9                    RET                  ;retour au BASIC
                           ORG  70FDH
                           LOAD $
70FD 00          TEMPGHOSTS: DB  0
70FE 00          DONEFLAG:   DB  0
70FF 00          NUMGHOSTS:  DB  0
7100 0000        MX:         DW  0
7102 0000        G1X:        DW  0
7104 0000        G2X:        DW  0
7106 0000        G3X:        DW  0
```

```
 97 7108 0000     G4X:        DW   0
 98 710A 0000     G5X:        DW   0
 99 710C 0000     G6X:        DW   0
100 710E 0000     G7X:        DW   0
101 7110 0000     G8X:        DW   0
102 7112 0000     G9X:        DW   0
103 7114 0000     G10X:       DW   0
104                           END
```

# LA BIBLIOTHÈQUE SYBEX

## OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks,*
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks,*
636 pages, Réf. 340

TECHNIQUES D'INTERFACE aux microprocesseurs
*par Austin Lesea et Rodnay Zaks,*
450 pages, Réf. 339

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F
*par Joël Poncet,*
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux,*
140 pages, Réf. 369

LA SOLUTION RS-232 *par Joe Campbell,*
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par Pierrick Bourgault,*
198 pages, Réf. 0119

## BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux,*
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices
*par Jean-Pierre Lamoitier,*
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui,*
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller,*
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE
*par Richard Mateosian,*
352 pages, Réf. 429

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC
*par David H. Ahl,*
204 pages, Réf. 247

FICHIERS EN BASIC *par Alan Simpson,*
256 pages, Réf. 0102

TECHNIQUES DE PROGRAMMATION EN BASIC
*par S. Crosmarie, M. Perron et D. Philippine*
152 pages, Réf. 0124

## PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux,*
496 pages, Réf. 330

LE PASCAL PAR LA PRATIQUE
*par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

## AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 360

## MICRO-ORDINATEURS

### ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES
*par Rodnay Zaks,*
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par Norbert Rimoux,*
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR
*par Georges Fagot-Barraly,*
192 pages, Réf. 420

### AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR
*par Georges Fagot-Barraly,*
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

AMSTRAD, GUIDE DU GRAPHISME *par James Wynford,*
208 pages, Réf. 0141

AMSTRAD CP/M 2.2 *par* *Anatole D'Hardencourt*,
248 pages, Réf. 0156

## APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II,
Tomes 1 et 2 *par* *Léopold Laurent*,
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par* *Stanley R. Trost*,
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
*par* *Douglas Hergert et Joseph T. Kalash*,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par* *Douglas Hergert*,
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par* *Rodnay Zaks*,
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
*par* *Joseph Caggiano*,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
*par* *Thomas Blackadar*,
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
*par* *Goulven Habasque*,
240 pages, Réf. 0099

INTRODUCTION A MAC PASCAL *par* *Pierre Le Beux*,
416 pages, Réf. 0145

## ATARI

JEUX EN BASIC SUR ATARI *par* *Paul Bunn*,
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par* *Rodnay Zaks*,
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par* *Thomas Blackadar*,
192 pages, Réf. 354

## ATMOS

JEUX EN BASIC SUR ATMOS *par* *Pierre Monsaut*,
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par* *Stanley R. Trost*,
180 pages, Réf. 372

## COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
*par* *Pierre Monsaut*,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
*par* *Rodnay Zaks*,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
*par* *Douglas Hergert*,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
*par* *J. Kascmer*,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
*par* *Stanley R. Trost*,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
*par* *Charles Platt*,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par* *Eric Ravis*,
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
*par* *Marty Dejonghe et Caroline Earhart*,
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
*par* *Gary Lippman*,
216 pages, Réf. 0100

## DRAGON

JEUX EN BASIC SUR DRAGON *par* *Pierre Monsaut*,
96 pages, Réf. 324

## EXL 100

EXL 100, JEUX D'ACTION *par* *Pierre Monsaut*,
96 pages, Réf. 0126

## GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
*par* *François Abella*,
208 pages, Réf. 264

## HECTOR

HECTOR JEUX D'ACTION *par* *Pierre Monsaut*,
96 pages, Réf. 388

## IBM

IBM PC EXERCICES EN BASIC *par* *Jean-Pierre Lamoitier*,
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
*par* *Joan Lasselle et Carol Ramsey*,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par* *Stanley R. Trost*,
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par* *Nelson Ford*,
320 pages, Réf. 357

GUIDE DU PC DOS *par* *Richard A. King*,
240 pages, Réf. 0013

## LASER

LASER JEUX D'ACTION *par* *Pierre Monsaut*,
96 pages, Réf. 371

## MO 5

MO 5 JEUX D'ACTION *par* *Pierre Monsaut*,
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par* *Rodnay Zaks*,
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par* *Stanley R. Trost*,
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
*par* *Georges Fagot-Barraly*,
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par* *Daniel Lebigre*,
272 pages, Réf. 0118

MO 5, STATIQUE, DYNAMIQUE, ELECTRONIQUE,

PROGRAMMES DE PHYSIQUE EN BASIC
*par Beaufils, Lamarche et Muggianu,*
240 pages, Réf. 0148

## MSX

MSX, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw,*
192 pages, Réf. 0132

MSX, PROGRAMMES EN LANGAGE MACHINE
*par Steeve Webb,*
112 pages, Réf. 0153

MSX, PROGRAMMATION EN ASSEMBLEUR
*par Georges Fagot-Barraly,*
216 pages, Réf. 0144

MSX, GUIDE DU BASIC *par Michel Laurent,*
264 pages, Réf. 0155

## ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

## SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
*par Michel Lhoir,*
2 tomes, Réf. 261-262

## SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 377

## SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
*par S.M. Gee,*
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

## TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
*par François Abella,*
160 pages, Réf. 303

## TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
*par Georges Fagot-Barraly,*
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly,*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
*par Jean-Pierre Lhoir,*
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 374

## TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
*par Léopold Laurent,*
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
*par Pierre Monsaut,*
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
*par Orson Kellog,*
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
*par Rodnay Zaks,*
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
*par Stanley R. Trost,*
160 pages, Réf. 413

## VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
*par G. O. Hamann,*
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

## VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0128

## ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

## MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 358

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 274

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800
*par Daniel-Jean David et Rodnay Zaks,*
374 pages, Réf. 327

PROGRAMMATION DU 6809
*par Rodnay Zaks et William Labiak,*
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
*par James W. Coffron,*
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088 *par François Retoreau,*
616 pages, Réf. 0093

## SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
*par Charles W. Grant et Jon Butah,*
308 pages, Réf. 365

GUIDE DE MS-DOS *par Richard A. King,*
360 pages, Réf. 0117

INTRODUCTION A UNIX *par John D. Halamka,*
240 pages, Réf. 0098

## APPLICATIONS ET LOGICIELS

INTRODUCTION AU TRAITEMENT DE TEXTE
*par Hal Glatzer,*
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par Julie Anne Arca,*
320 pages, Réf. 0005

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson,*
280 pages, Réf. 0064

DE VISICALC A VISI ON *par Jacques Bourdeu,*
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
*par D. Helle et G. Boussand,*
304 pages, Réf. 0079

dBASE II APPLICATIONS *par Christophe Stehly,*
248 pages, Réf. 416

INTRODUCTION A LOTUS 1-2-3
*par Chris Gilbert et Laurie Williams,*
272 pages, Réf. 0106

LOGISTAT, ANALYSE STATISTIQUE DES DONNÉES
*par Fredj Tekaia et Michele Bidel,*
352 pages, Réf. 0132

**La plupart de ces ouvrages existent en version anglaise.**

## EN ANGLAIS

BASIC EXERCISES FOR APPLE *by Jean-Pierre Lamoitier,*
232 pages, Réf. 0-084

BASIC FOR BUSINESS *by Douglas Hergert,*
224 pages, Réf. 0-080

CELESTIAL BASIC : Astronomy on your Computer
*by Eric Burgess,*
228 pages, Réf. 0-087

INTRODUCTION TO PASCAL (Including UCSD Pascal)
*by Rodnay Zaks,*
422 pages, Réf. 0-066

DOING BUSINESS WITH PASCAL
*by Richard Hergert and Douglas Hergert,*
380 pages, Réf. 0-091

MASTERING VISICALC *by Douglas Hergert,*
224 pages, Réf. 0-090

THE APPLE CONNECTION *by James W. Coffron,*
228 pages, Réf. 0-085

PROGRAMMING THE Z8000 *by Richard Mateosian,*
300 pages, Réf. 0-032

A MICROPROGRAMMED APL IMPLEMENTATION
*by Rodnay Zaks,*
350 pages, Réf. 0-005

ADVANCED 6502 PROGRAMMING *by Rodnay Zaks,*
292 pages, Réf. 0-089

FORTRAN PROGRAMS FOR SCIENTISTS AND ENGINEERS *by Alan R. Miller,*
320 pages, Réf. 0-082

---

## *POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS*


FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 42.03.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 626441
Telex : 08588163

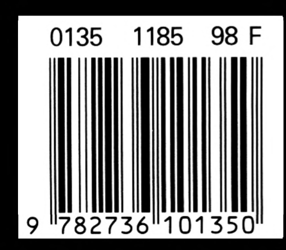




**Paris • Berkeley • Düsseldorf**

Achevé d'imprimer le 26 novembre 1985 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 4e trimestre 1985 - Imprimeur n° 1908

L'Amstrad CPC464 possède l'un des meilleurs BASIC actuellement disponibles. A ceux qui ont déjà acquis les rudiments du BASIC, ce livre permettra d'approfondir leur connaissance de la programmation tout en découvrant les particularités qui font de l'Amstrad un des ordinateurs les plus puissants de sa catégorie. Après une première partie consacrée à la présentation de la machine, on étudiera en détail les possibilités graphiques et sonores du BASIC Amstrad. Trois chapitres concernent l'utilisation de l'assembleur ZEN et décrivent la programmation en assembleur, l'interfaçage avec le BASIC et l'utilisation des routines du système d'exploitation. De nombreux exemples de programmes viennent illustrer chaque sujet.




SYBEX
L'AMSTRAD EXPLORÉ


AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
AC
ME
https://acpc.me/